VectSharp

2.5.0

Generated by Doxygen 1.9.5

Chapter 1

VectSharp: a light library for C# vector graphics

1.1 Introduction

VectSharp is a library to create vector graphics (including text) in C#, without too many dependencies.

VectSharp is written using .NET Core, and is available for Mac, Windows and Linux. Since version 2.0.0, it is released under an LGPLv3 license. It includes 14 standard fonts, originally released under an ASL-2.0 license.

It includes an abstract layer on top of which output layers can be written. Currently, there are five available output layers:

- **VectSharp.PDF** produces PDF documents.
- **VectSharp.Canvas** produces an `Avalonia.Controls.Canvas` object (https://avaloniaui.net/docs/controls/canvas) containing the rendered graphics objects.
- **VectSharp.SVG** produces vector graphics in SVG format.
- **VectSharp.Raster** produces raster images in PNG format, (this is done by rendering the image to a PDF document, and then using the `MuPDFCore` library to render the PDF). Since version 2.0.0, VectSharp.Raster is released under an AGPLv3 licence.
- **VectSharp.Raster.ImageSharp** produces raster images in multiple formats (BMP, GIF, JPEG, PBM, PNG, TGA, TIFF, WebP) using the `SixLabors.ImageSharp` library.

VectSharp.Raster and VectSharp.Raster.ImageSharp are somewhat overlapping, as both of them can be used to create PNG images. However, VectSharp.Raster is much faster, though it only supports the PNG format. Instead, VectSharp.Raster.ImageSharp is slower, but supports more formats and has a more permissive licence. Another difference is that VectSharp.Raster carries a native dependency (through MuPDFCore), while VectSharp.ImageSharp does not.

Furthermore:

- The **`VectSharp.Plots`** package contains classes and methods to draw plots (such as scatter plots, line charts, bar charts, box plots, function plots, and more).
- **`VectSharp.ThreeD`** adds support for 3D vector and raster graphics.

- **VectSharp.Markdown** can be used to transform Markdown documents into VectSharp objects, that can then be exported e.g. as PDF or SVG files, or displayed in an Avalonia `Canvas`. **VectSharp.MarkdownCanvas** uses VectSharp.Markdown to render Markdown documents in Avalonia applications (an example of this is in the `MarkdownViewerDemo project`).
- **VectSharp.MuPDFUtils**, released under an AGPLv3 licence, contains some utility functions that use `MuPDFCore` to make it possible to include in VectSharp graphics images in various formats.
- **VectSharp.ImageSharpUtils** adds the same capabilities as VectSharp.MuPDFUtils, using ImageSharp instead of MuPDFCore; as a result, it is released under a more permissive LGPLv3 licence.
- **VectSharp.Fonts.Nimbus** is a package released under a GPLv3 license, which contains the standard fonts that were used in VectSharp before version 2.0.0. Since these fonts are released under a GPL license, they had to be replaced when the VectSharp license changed to LGPL. See the [Font libraries](section) below for information on how to re-enable these fonts.
- The **Animation** class (provided in the base VectSharp package) can be used to create animations that can be saved as animated GIFs (using VectSharp.Raster.ImageSharp), SVGs (using VectSharp.SVG) and PNGs (using VectSharp.Raster or VectSharp.Raster.ImageSharp).

1.2 Installing VectSharp

To include VectSharp in your project, you will need one of the output layer NuGet packages: `VectSharp.PDF`, `VectSharp.Canvas`, `VectSharp.Raster`, `VectSharp.Raster.ImageSharp`, or `VectSharp.SVG`. You will need `VectSharp.ThreeD` to work with 3D graphics or `VectSharp.Plots` to create plots. You may want the `VectSharp.MuPDFUtils` package if you wish to manipulate raster images, and the `VectSharp.Fonts.Nimbus` if you want to restore the GPL-licensed fonts used in previous versions of the library.

1.3 Usage

You can find detailed documentation for the VectSharp library, **including interactive examples**, at the `documentation website`. A comprehensive API reference is also available, both as a `website` and as a `PDF manual`.

In general, working with VectSharp involves: creating a `Document`, adding `Page`s, drawing to the `Page`s' `Graphics` objects and, finally, exporting them to a PDF document, `Canvas`, PNG image or SVG document.

- Create a `Document`:
```
using VectSharp;
// ...
Document doc = new Document();
```
- Add a `Page`:
```
doc.Pages.Add(new Page(1000, 1000));
```
- Draw to the `Page`'s `Graphics` object:
```
Graphics gpr = doc.Pages.Last().Graphics;
gpr.FillRectangle(100, 100, 800, 800, Colour.FromRgb(128, 128, 128));
```
- Save as PDF document:
```
using VectSharp.PDF;
//...
doc.SaveAsPDF(@"Sample.pdf");
```
- Export the graphics to a `Canvas`:
```
using VectSharp.Canvas;
//...
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToCanvas();
```

- Export the graphics to a `Canvas`, using a multi-layer, multi-threaded, triple-buffered renderer based on SkiaSharp (which provides the best performance if you wish e.g. to place the canvas within a `Zoom↩Border`):
```
using VectSharp.Canvas;
//...
// A single page
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToSKCanvas();
// The whole document - each page will correspond to a layer
Avalonia.Controls.Canvas can = doc.PaintToSKCanvas();
```

- Save as a PNG image:
```
using VectSharp.Raster;
//...
doc.Pages.Last().SaveAsPNG(@"Sample.png");
```

- Save as a JPEG image:
```
using VectSharp.Raster.ImageSharp;
//...
doc.Pages.Last().SaveAsImage(@"Sample.jpg");
```

- Save as an SVG document:
```
using VectSharp.SVG;
//...
doc.Pages.Last().SaveAsSVG(@"Sample.svg");
```

- PDF and SVG documents support both internal and external links:
```
using VectSharp;
using VectSharp.PDF;
using VectSharp.SVG;
//...
Document document = new Document();
Page page = new Page(1000, 1000);
document.Pages.Add(page);
page.Graphics.FillRectangle(100, 100, 800, 50, Colour.FromRgb(128, 128, 128), tag:  "linkToGitHub");
page.Graphics.FillRectangle(100, 300, 800, 50, Colour.FromRgb(255, 0, 0), tag:
"linkToBlueRectangle");
page.Graphics.FillRectangle(100, 850, 800, 50, Colour.FromRgb(0, 0, 255), tag:  "blueRectangle");
Dictionary<string, string> links = new Dictionary<string, string>() { { "linkToGitHub",
"https://github.com/" }, { "linkToBlueRectangle", "#blueRectangle" } };
page.SaveAsSVG(@"Links.svg", linkDestinations:  links);
document.SaveAsPDF(@"Links.pdf", linkDestinations:  links);
```

  This code produces a document with three rectangles: the grey one at the top links to the GitHub home page, while the red one in the middle is a hyperlink to the blue one at the bottom. Links in PDF documents can refer to objects that are in a different page than the one containing the link.

The public classes and methods are `fully documented` (with interactive examples created using Blazor), and you can find a (much) more detailed code example in `MainWindow.xaml.cs`. A detailed guide about 3D graphics in VectSharp.ThreeD is available in the `VectSharp.ThreeD` folder. Further example code for animations is available in the `DemoAnimation project`.

1.4 Font libraries

Since version 2.0.0, font names are resolved using a "font library". This is a class that implements the `VectSharp.IFontLibrary` interface, providing methods to obtain a `FontFamily` object from a `string` or a `FontFamily.StandardFontFamilies` enumeration. The default font library included in VectSharp uses the embedded fonts (Arimo, Tinos, Cousine) as the standard font families.

In practice, assuming you want to use the default font library, you have the following options to create a `Font↩Family` object:
```
using VectSharp;
// ...
FontFamily helvetica = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Arimo font family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Tinos font family.
```

These replace the `FontFamily(string)` and `FontFamily(StandardFontFamilies)` constructors of previous versions of VectSharp. Overloads of this method let you specify a list of "fallback" fonts that will be used if the first font you specify is not available.

If you wish, you can replace the default font library with a different one; this will change the way font families are resolved. For example, after installing the `VectSharp.Fonts.Nimbus` NuGet package, you can do:

```
using VectSharp;
// ...
FontFamily.DefaultFontLibrary = VectSharp.Fonts.Nimbus.Library;
FontFamily helvetica = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Nimbus Sans L font family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Nimbus Roman No 9 L
      font family.
```

This will let you re-enable the fonts that were used in previous versions of VectSharp.

You can also use multiple font libraries in the same project. Again, assuming you have installed the VectSharp.Fonts.Nimbus NuGet package:

```
using VectSharp;
FontFamily helvetica1 = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Arimo font family.
FontFamily times1 = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Tinos font family.
FontFamily helvetica2 = VectSharp.Fonts.Nimbus.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
      // Will resolve to the Nimbus Sans L font family.
FontFamily times2 = VectSharp.Fonts.Nimbus.ResolveFontFamily("Times-Roman"); // Will resolve to the Nimbus
      Roman No 9 L font family.
```

Finally, you can create your own font library class (which could implement things such as dowloading fonts from Google Fonts, or finding them in the user's system font directory...) by creating a class that implements the `IFont↩Library` interface or that extends the `FontLibrary` class (in this latter case, you get a default implementation for the `ResolveFontFamily` overloads that use a list of fallback fonts).

1.5 Creating new output layers

VectSharp can be easily extended to provide additional output layers. To do so:

1. Create a new class implementing the `IGraphicsContext` interface.
2. Provide an extension method to either the `Page` or `Document` types.
3. Somewhere in the extension method, call the `CopyToIGraphicsContext` method on the `Graphics` object of the `Page`s.
4. Opportunely save or return the rendered result.

1.6 Compiling VectSharp from source

The VectSharp source code includes an example project (*VectSharp.Demo*) presenting how VectSharp can be used to produce graphics.

To be able to compile VectSharp from source, you will need to install the latest `.NET SDK` for your operating system.

You can use `Microsoft Visual Studio` to compile the program. The following instructions will cover compiling VectSharp from the command line, instead.

First of all, you will need to download the VectSharp source code: `VectSharp.tar.gz` and extract it somewhere.

1.6.1 Windows

Open a command-line window in the folder where you have extracted the source code, and type:

```
BuildDemo <Target>
```

Where `<Target>` can be one of `Win-x64`, `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

1.6.2 macOS and Linux

Open a terminal in the folder where you have extracted the source code, and type:

```
./BuildDemo.sh <Target>
```

Where `<Target>` can be one of `Win-x64`, `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

If you receive an error about permissions being denied, try typing `chmod +x BuildDemo.sh` first.

1.7 Note about VectSharp.MuPDFUtils and .NET Framework

If you wish to use VectSharp.MuPDFUtils in a .NET Framework project, you will need to manually copy the native MuPDFWrapper library for the platform you are using to the executable directory (this is done automatically if you target .NET core).

One way to obtain the appropriate library files is:

1. Manually download the NuGet package for `MuPFDCore` (click on the "Download package" link on the right).
2. Rename the `.nupkg` file so that it has a `.zip` extension.
3. Extract the zip file.
4. Within the extracted folder, the library files are in the `runtimes/xxx-yyy/native/` folder, where `xxx` is either `linux`, `osx` or `win`, depending on the platform you are using, and `yyy` is `x64`, `x86` or `arm64` depending on the architecture.

Make sure you copy the appropriate file to the same folder as the executable!

Chapter 2

Namespace Index

2.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 VectSharp Namespace Reference

Classes

- class AnimatedPNG

  Contains methods to create animated PNG image files.
- class Animation

  Describes an animation constituted by a number of frames and transitions between them.
- class Brush

  Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.
- struct Colour

  Represents an RGB colour.
- class Colours

  Standard colours.
- class DefaultFontLibrary

  A default font library that resolves standard families using the embedded fonts.
- class DisposableIntPtr

  An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.
- class Document

  Represents a collection of pages.
- class FolderFontLibrary

  A font library that resolves fonts from a folder containing TrueType files.
- class Font

  Represents a typeface with a specific size.
- class FontFamily

  Represents a typeface.
- class FontFamilyCreationException

  An exception that occurs while creating a FontFamily.
- class FontLibrary

  Abstract class with a default implementation of font family fallbacks.
- class FormattedText

  Represents a run of text that should be drawn with the same style.
- class FormattedTextExtensions

  Contains extension methods for collections of FormattedText objects.

- class Frame

  A key frame for an animation.
- class GradientBrush

  Represents a brush painting with a gradient.
- class Gradients

  Standard gradients.
- struct GradientStop

  Represents a colour stop in a gradient.
- class GradientStops

  Represents a read-only list of GradientStops.
- class Graphics

  Represents an abstract drawing surface.
- class GraphicsPath

  Represents a graphics path that can be filled or stroked.
- interface IEasing

  Describes a function used to transform the transition speed.
- interface IFontLibrary

  Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.
- interface IGraphicsContext

  This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.
- class LinearEasing

  Describes a linear easing (i.e., no easing).
- class LinearGradientBrush

  Represents a brush painting with a linear gradient.
- struct LineDash

  Represents instructions on how to paint a dashed line.
- class MultiFontLibrary

  A font library that tries to resolve fonts using other font libraries.
- class Page

  Represents a Graphics object with a width and height.
- struct Point

  Represents a point relative to an origin in the top-left corner.
- class RadialGradientBrush

  Represents a brush painting with a radial gradient.
- class RasterImage

  Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.
- struct Rectangle

  Represents a rectangle.
- class ResourceFontFamily

  Represents a FontFamily created from a resource stream.
- class Segment

  Represents a segment as part of a GraphicsPath.
- class SimpleFontLibrary

  A font library that can be used to cache and resolve font family names.
- struct Size

  Represents the size of an object.
- class SolidColourBrush

  Represents a brush painting with a single solid colour.
- class SplineEasing

  Describes an easing defined by a Cubic Bezier curve.

- class Transition

  Describes the transition between two successive Frames.
- class TrueTypeFile

  Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.microsoft.com/en-us/typography/opentype/spec/
- class UnbalancedStackException

  The exception that is thrown when an unbalanced graphics state stack occurs.

Enumerations

- enum TextBaselines

  Represent text baselines.
- enum TextAnchors

  Represents text anchors.
- enum LineCaps

  Represents line caps.
- enum LineJoins

  Represents line joining options.
- enum SegmentType

  Types of Segment.
- enum UnbalancedStackActions

  Represents ways to deal with unbalanced graphics state stacks.
- enum Script

  Represents the position of the text.
- enum FillRule

  Represents a rule used to determine whether a point is inside or outside of a shape.
- enum PixelFormats

  Represents the pixel format of a raster image.

6.1.1 Enumeration Type Documentation

6.1.1.1 FillRule

`enum VectSharp.FillRule`

Represents a rule used to determine whether a point is inside or outside of a shape.

Definition at line 262 of file Graphics.cs.

6.1.1.2 LineCaps

`enum VectSharp.LineCaps`

Represents line caps.

Definition at line 70 of file Enums.cs.

6.1.1.3 LineJoins

enum VectSharp.LineJoins

Represents line joining options.

Definition at line 91 of file Enums.cs.

6.1.1.4 PixelFormats

enum VectSharp.PixelFormats

Represents the pixel format of a raster image.

Definition at line 27 of file RasterImage.cs.

6.1.1.5 Script

enum VectSharp.Script

Represents the position of the text.

Definition at line 29 of file FormattedText.cs.

6.1.1.6 SegmentType

enum VectSharp.SegmentType

Types of Segment.

Definition at line 151 of file Enums.cs.

6.1.1.7 TextAnchors

enum VectSharp.TextAnchors

Represents text anchors.

Definition at line 49 of file Enums.cs.

6.1.1.8 TextBaselines

`enum VectSharp.TextBaselines`

Represent text baselines.

Definition at line 23 of file Enums.cs.

6.1.1.9 UnbalancedStackActions

`enum VectSharp.UnbalancedStackActions`

Represents ways to deal with unbalanced graphics state stacks.

Definition at line 182 of file Enums.cs.

6.2 VectSharp.Canvas Namespace Reference

Classes

- class AnimatedCanvas

  An Avalonia.Controls.Canvas containing an animation.
- class AvaloniaContextInterpreter

  Contains methods to render a Page to an Avalonia.Controls.Canvas.
- class FilterOption

  Determines how and whether image filters are rasterised.
- class RenderAction

  Represents a light-weight rendering action.
- class SKMultiLayerRenderCanvas

  Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.
- class SKRenderAction

  Represents a light-weight rendering action.
- class SKRenderContext

  Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.
- class SKRenderContextInterpreter

  Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

6.3 VectSharp.Filters Namespace Reference

Classes

- class BoxBlurFilter

  Represents a filter applying a box blur.
- class ColourMatrix

  Represents a colour transformation matrix.
- class ColourMatrixFilter

  Represents a filter that applies a Filters.ColourMatrix to the colours of the image.
- class CompositeLocationInvariantFilter

  Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.
- class ConvolutionFilter

  Represents a filter that applies a matrix convolution to the image.
- class FilterWithRasterisableParameter

  Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.
- class GaussianBlurFilter

  Represents a filter that applies a Gaussian blur effect.
- interface IFilter

  Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.
- interface IFilterWithLocation

  Represents a filter whose results depend on the position of the subject image on the graphics surface.
- interface IFilterWithRasterisableParameter

  Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.
- interface ILocationInvariantFilter

  Represents a filter that can be applied to an image regardless of its location on the graphics surface.
- class MaskFilter

  Represents a filter that uses the luminance of an image to mask another image.

6.4 VectSharp.Fonts Namespace Reference

Classes

- class Nimbus

  Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

6.5 VectSharp.ImageSharpUtils Namespace Reference

Classes

- class ImageURIParser

  Provides a method to parse an image URI into a page.
- class RasterImageFile

  A RasterImage created from a file.
- class RasterImageStream

  A RasterImage created from a stream.

6.6 VectSharp.Markdown Namespace Reference

Classes

- struct FormattedString

  Represents a string with associated formatting information.
- class HTTPUtils

  Contains utilities to resolve absolute and relative URIs.
- class Margins

  Represents the margins of a page.
- class MarkdownRenderer

  Renders Markdown documents into VectSharp graphics objects.
- class SyntaxHighlighter

  Contains methods to perform syntax highlighting.

6.7 VectSharp.MarkdownCanvas Namespace Reference

Classes

- class MarkdownCanvasControl

  A control to display a Markdown document in an Avalonia application.

6.8 VectSharp.MuPDFUtils Namespace Reference

Classes

- class ImageURIParser

  Provides a method to parse an image URI into a page.
- class RasterImageFile

  A RasterImage created from a file.
- class RasterImageStream

  A RasterImage created from a stream.

6.9 VectSharp.PDF Namespace Reference

Classes

- class PDFContextInterpreter

  Contains methods to render a Document as a PDF document.

6.10 VectSharp.Plots Namespace Reference

Classes

- class ActionDataPointElement

  A symbol drawn by a custom Action.
- class Area

  A plot element that fills an area between a line passing through some data points and a base line.
- class Bars

  A plot element that draws bars.
- class BoxPlot

  A plot element that draws a box plot.
- class CategoricalBars

  A plot element that draws bars for categorical data.
- class CategoricalCoordinateSystem1D

  Represents a categorical 1-D coordinate system.
- class ClusteredBars

  A plot element that draws clusters of bars.
- class CompositeCoordinateSystem2D

  Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.
- class ContinuousAxis

  A plot element that draws an axis line and an arrow.
- class ContinuousAxisLabels

  A plot element that draws equally spaced labels on an axis.
- class ContinuousAxisTicks

  A plot element that draws equally spaced ticks on an axis.
- class ContinuousAxisTitle

  A plot element that draws a title for an axis.
- class CoordinateSystem

  A coordinate system using a custom method to transform data points.
- class CoordinateSystem1D

  A coordinate system using a custom method to transform data points.
- class DataLabels

  A plot element that draws a text label at each data point.
- class DataLine

  A plot element that draws a line passing through a set of points.
- class ExponentialTrendLine

  A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.
- class Function2D

  A plot element that plots a function of two variables.
- class Function2DGrid

  Represents a function of two variables that has been sampled in some points.
- class GraphicsDataPointElement

  A symbol defined by a VectSharp.Graphics object.
- class Grid

  A plot element that draws a grid.
- interface IContinuousCoordinateSystem

  Represents a coordinate system performing continuous transformations.
- interface IContinuousInvertibleCoordinateSystem

  Represents a coordinate system performing continuous invertible transformations.

- interface ICoordinateSystem

  Represents a coordinate system.
- interface ICoordinateSystem1D

  Represents a coordinate system tranforming data points of type T into `doubles`.
- interface IDataPointElement

  Represents a symbol that can be added to the plot at a specified position.
- interface IPlotElement

  Represents a plot element.
- class LinearCoordinateSystem1D

  Represents a 1-D linear coordinate system.
- class LinearCoordinateSystem2D

  Represents a linear coordinate system.
- class LinearTrendLine

  A plot element that draws a linear trendline with equation `y = a * x + b`.
- class LinLogCoordinateSystem2D

  Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.
- class LogarithmicCoordinateSystem1D

  Represents a 1-D logarithmic coordinate system.
- class LogarithmicCoordinateSystem2D

  Represents a logarithmic coordinate system.
- class LogarithmicTrendLine

  A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.
- class LogLinCoordinateSystem2D

  Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.
- class MovingAverageTrendLine

  A plot element that draws a moving average trendline.
- class PathDataPointElement

  A symbol defined by a GraphicsPath.
- class Pie

  A plot element that draws a pie or a doughnut.
- class Plot

  Represents a collection of plot elements.
- class PlotElement

  A plot element that uses an Action to draw its contents.
- class PlotElementPresentationAttributes

  Determines the appearance of plot elements.
- class PolynomialTrendLine

  A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.
- class PowerLawTrendLine

  A plot element that draws a power law trendline with equation `y = b * x^a`.
- class ScatterPoints

  A plot element that draws a symbol at the location of multiple data points.
- class StackedBars

  A plot element that draws stacked bars.
- class TextLabel

  A plot element that draws a single text label.
- class Violin

  A plot element that draws a violin plot.

6.11 VectSharp.Raster Namespace Reference

Classes

- class Raster

  Contains methods to render a page to a PNG image.

6.12 VectSharp.Raster.ImageSharp Namespace Reference

Classes

- class ImageSharpContextInterpreter

  Contains methods to render a Page to an Image.

Enumerations

- enum OutputFormats

  Enumeration containing the supported output formats.

6.12.1 Enumeration Type Documentation

6.12.1.1 OutputFormats

`enum VectSharp.Raster.ImageSharp.OutputFormats`

Enumeration containing the supported output formats.

Definition at line 1004 of file ImageSharpContext.cs.

6.13 VectSharp.SVG Namespace Reference

Classes

- class Parser

  Contains methods to read an SVG image file.
- class SVGContextInterpreter

  Contains methods to render a Page as an SVG file.

6.14 VectSharp.ThreeD Namespace Reference

Classes

- class AmbientLightSource

  Represents a uniform ambien light source.
- class AreaLightSource

  Represents a light source emitting light from a circular area.
- class ColourMaterial

  Represents a material that always has the same colour, regardless of light.
- interface ILightSource

  Represents a light source.
- interface IMaterial

  Represents a material used to the determine the appearance of Triangle3DElement.
- interface IScene

  Represents a 3D scene.
- struct LightIntensity

  Represents the intensity of a light source at a particular point.
- class MaskedLightSource

  Represents a point light source with a stencil in front of it.
- class ObjectFactory

  A static class containing methods to create complex 3D objects.
- class ParallelLightSource

  Represents a parallel light source.
- class PhongMaterial

  Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.
- class PointLightSource

  Represents a point light source.
- class Scene

  Represents a 3D scene.
- class SpotlightLightSource

  Represents a conic spotlight.

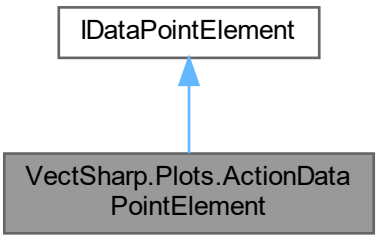
Chapter 7

Class Documentation

7.1 VectSharp.Plots.ActionDataPointElement Class Reference

A symbol drawn by a custom Action.

Inheritance diagram for VectSharp.Plots.ActionDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- ActionDataPointElement (Action< Graphics, PlotElementPresentationAttributes, string > plotAction)

  Creates a new ActionDataPointElement using the specified action to draw the symbol.

Properties

- Action< Graphics, PlotElementPresentationAttributes, string > PlotAction `[get, set]`

  The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

7.1.1 Detailed Description

A symbol drawn by a custom Action.

Definition at line 122 of file DataPoints.cs.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ActionDataPointElement()

```
VectSharp.Plots.ActionDataPointElement.ActionDataPointElement (
            Action< Graphics, PlotElementPresentationAttributes, string > plotAction )
```

Creates a new ActionDataPointElement using the specified action to draw the symbol.

Parameters

plotAction	The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

Definition at line 140 of file DataPoints.cs.

7.1.3 Member Function Documentation

7.1.3.1 Plot()

```
void VectSharp.Plots.ActionDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

7.1.4 Property Documentation

7.1.4.1 PlotAction

```
Action<Graphics, PlotElementPresentationAttributes, string> VectSharp.Plots.ActionDataPoint↩
Element.PlotAction [get], [set]
```

The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

Definition at line 129 of file DataPoints.cs.

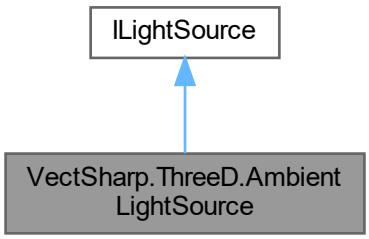
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.2 VectSharp.ThreeD.AmbientLightSource Class Reference

Represents a uniform ambien light source.

Inheritance diagram for VectSharp.ThreeD.AmbientLightSource:



Public Member Functions

- AmbientLightSource (double intensity)

  Creates a new AmbientLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- double Intensity `[get, set]`

  The intensity of the light.
- bool CastsShadow `[get]`

  Determines whether the light casts a shadow or not.

7.2.1 Detailed Description

Represents a uniform ambien light source.

Definition at line 91 of file Lights.cs.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 AmbientLightSource()

```
VectSharp.ThreeD.AmbientLightSource.AmbientLightSource (
            double intensity )
```

Creates a new AmbientLightSource instance.

Parameters

intensity	The intensity of the light.

Definition at line 105 of file Lights.cs.

7.2.3 Member Function Documentation

7.2.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.AmbientLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 111 of file Lights.cs.

7.2.3.2 GetObstruction()

```
double VectSharp.ThreeD.AmbientLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 117 of file Lights.cs.

7.2.4 Property Documentation

7.2.4.1 CastsShadow

```
bool VectSharp.ThreeD.AmbientLightSource.CastsShadow  [get]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 99 of file Lights.cs.

7.2.4.2 Intensity

```
double VectSharp.ThreeD.AmbientLightSource.Intensity  [get], [set]
```

The intensity of the light.

Definition at line 96 of file Lights.cs.

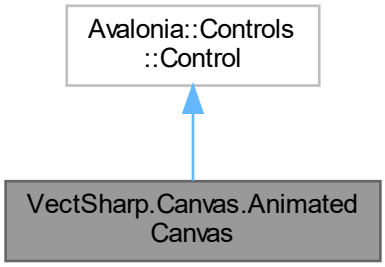
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.3 VectSharp.Canvas.AnimatedCanvas Class Reference

An Avalonia.Controls.Canvas containing an animation.

Inheritance diagram for VectSharp.Canvas.AnimatedCanvas:



Public Member Functions

- override void Render (DrawingContext context)

Static Public Attributes

- static readonly StyledProperty< int > CurrentFrameProperty

  Defines the CurrentFrame property.
- static readonly DirectProperty< AnimatedCanvas, int > FrameCountProperty = AvaloniaProperty.Register↩ Direct<AnimatedCanvas, int>(nameof(FrameCount), o => o.maxFrameIndex)

  Defines the FrameCount property.
- static readonly DirectProperty< AnimatedCanvas, double > FrameRateProperty = AvaloniaProperty.↩ RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o => o.FrameRate)

  Defines the FrameRate property.
- static readonly StyledProperty< bool > IsPlayingProperty

  Defines the IsPlaying property.

Properties

- int CurrentFrame `[get, set]`

  The current frame in the animation.
- int FrameCount `[get]`

  The number of frames in the animation.
- double FrameRate `[get]`

  The target frame rate of the animation.
- bool IsPlaying `[get, set]`

  The current frame in the animation.

7.3.1 Detailed Description

An Avalonia.Controls.Canvas containing an animation.

Definition at line 28 of file AnimatedCanvas.cs.

7.3.2 Member Function Documentation

7.3.2.1 Render()

```
override void VectSharp.Canvas.AnimatedCanvas.Render (
            DrawingContext context )
```

Definition at line 203 of file AnimatedCanvas.cs.

7.3.3 Member Data Documentation

7.3.3.1 CurrentFrameProperty

```
readonly StyledProperty<int> VectSharp.Canvas.AnimatedCanvas.CurrentFrameProperty  [static]
```

Initial value:

```
= AvaloniaProperty.Register<AnimatedCanvas, int>(nameof(CurrentFrame), coerce:  (ownerObject, val) =>
        {
            AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
            if (owner.maxFrameIndex > 0)
            {
                if (val < owner.maxFrameIndex)
                {
                    return val;
                }
                else
                {
                    owner.IsPlaying = false;
                    return owner.maxFrameIndex - 1;
                }
            }
            else
            {
                return val % owner.RenderActions.Length;
            }
        })
```

Defines the CurrentFrame property.

Definition at line 33 of file AnimatedCanvas.cs.

7.3.3.2 FrameCountProperty

```
readonly DirectProperty<AnimatedCanvas, int> VectSharp.Canvas.AnimatedCanvas.FrameCount↩
Property = AvaloniaProperty.RegisterDirect<AnimatedCanvas, int>(nameof(FrameCount), o => o.↩
maxFrameIndex)  [static]
```

Defines the FrameCount property.

Definition at line 67 of file AnimatedCanvas.cs.

7.3.3.3 FrameRateProperty

```
readonly DirectProperty<AnimatedCanvas, double> VectSharp.Canvas.AnimatedCanvas.FrameRate↩
Property = AvaloniaProperty.RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o =>
o.FrameRate)  [static]
```

Defines the FrameRate property.

Definition at line 91 of file AnimatedCanvas.cs.

7.3.3.4 IsPlayingProperty

```
readonly StyledProperty<bool> VectSharp.Canvas.AnimatedCanvas.IsPlayingProperty  [static]
```

Initial value:

```
= AvaloniaProperty.Register<AnimatedCanvas, bool>(nameof(IsPlaying), false, coerce:  (ownerObject, val) =>
        {
            AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
            if (owner.IsPlaying && !val)
            {
                owner.RefreshTimer.Dispose();
            }
            else if (!owner.IsPlaying && val)
            {
                owner.RefreshTimer = new System.Threading.Timer(_ =>
                {
                    _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
                    {
                        owner.CurrentFrame++;
                    });
                }, null, 0, (long)(1000.0 / owner.FrameRate));
            }
            return val;
        })
```

Defines the IsPlaying property.

Definition at line 115 of file AnimatedCanvas.cs.

7.3.4 Property Documentation

7.3.4.1 CurrentFrame

```
int VectSharp.Canvas.AnimatedCanvas.CurrentFrame  [get], [set]
```

The current frame in the animation.

Definition at line 58 of file AnimatedCanvas.cs.

7.3.4.2 FrameCount

```
int VectSharp.Canvas.AnimatedCanvas.FrameCount  [get]
```

The number of frames in the animation.

Definition at line 74 of file AnimatedCanvas.cs.

7.3.4.3 FrameRate

```
double VectSharp.Canvas.AnimatedCanvas.FrameRate  [get]
```

The target frame rate of the animation.

Definition at line 98 of file AnimatedCanvas.cs.

7.3.4.4 IsPlaying

```
bool VectSharp.Canvas.AnimatedCanvas.IsPlaying  [get], [set]
```

The current frame in the animation.

Definition at line 140 of file AnimatedCanvas.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/AnimatedCanvas.cs

7.4 VectSharp.AnimatedPNG Class Reference

Contains methods to create animated PNG image files.

Classes

- class CompressedFrame

  Represents an individual frame of a PNG animation.

Public Types

- enum InterframeCompression

  Types of inter-frame compression.

Static Public Member Functions

- static unsafe void Create (Stream outputStream, int width, int height, bool hasAlpha, IReadOnlyList< CompressedFrame > compressedFrames, int repeatCount)

  Create a new animated PNG image, outputting it to the specified stream.

7.4.1 Detailed Description

Contains methods to create animated PNG image files.

Definition at line 30 of file ImageFormats.cs.

7.4.2 Member Enumeration Documentation

7.4.2.1 InterframeCompression

```
enum VectSharp.AnimatedPNG.InterframeCompression
```

Types of inter-frame compression.

Definition at line 35 of file ImageFormats.cs.

7.4.3 Member Function Documentation

7.4.3.1 Create()

```
static unsafe void VectSharp.AnimatedPNG.Create (
            Stream outputStream,
            int width,
            int height,
            bool hasAlpha,
            IReadOnlyList< CompressedFrame > compressedFrames,
            int repeatCount )  [static]
```

Create a new animated PNG image, outputting it to the specified stream.

Parameters

outputStream	The stream to which the animated PNG image will be written.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the frames of the image have an alpha channel, set this to `true`; otherwise, set it to `false`.
compressedFrames	The frames that will be used to create the animated PNG image.
repeatCount	The number of times that the animation should loop. Set this to 0 for an infinitely repeating animation.

Definition at line 181 of file ImageFormats.cs.

The documentation for this class was generated from the following file:

- VectSharp/ImageFormats.cs

7.5 VectSharp.Animation Class Reference

Describes an animation constituted by a number of frames and transitions between them.

Public Member Functions

- Animation (double width, double height, double linearisationResolution)

  Creates a new Animation with the specified width, height and linearisation resolution.
- Page GetFrameAtAbsolute (double time)

  Obtains the (interpolated) frame that should be displayed after the specified time has passed since the start of the animation.
- Page GetFrameAtRelative (double relativeTime)

  Obtains the (interpolated) frame that should be displayed after the specified relative time has passed since the start of the animation.
- void AddFrame (Frame frame, Transition transition=null)

  Adds a new frame to the animation, with the specified transition.
- void RemoveLastFrame ()

  Removes the last frame from the animation (and the corresponding transition).

Properties

- ImmutableList< Frame > Frames = ImmutableList<Frame>.Empty `[get]`

  The key frames of the animation.
- ImmutableList< Transition > Transitions = ImmutableList<Transition>.Empty `[get]`

  The transitions between successive frames of the animation. This array always contains one fewer element than Frames.
- double Width `[get, set]`

  The width of the animation.
- double Height `[get, set]`

  The height of the animation.
- Colour Background = Colour.FromRgba(255, 255, 255, 0) `[get, set]`

  The background colour of the animation.
- double LinearisationResolution `[get]`

  The absolute length between successive samples to use when linearising GraphicsPaths.
- double Duration = 0 `[get]`

  The total duration of the animation (not including the number of repeats).
- int RepeatCount = 0 `[get, set]`

  The number of times that the animation should repeat.

7.5.1 Detailed Description

Describes an animation constituted by a number of frames and transitions between them.

Definition at line 1380 of file Animation.cs.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Animation()

```
VectSharp.Animation.Animation (
            double width,
            double height,
            double linearisationResolution )
```

Creates a new Animation with the specified width, height and linearisation resolution.

Parameters

width	The width of the animation.
height	The height of the animation.
linearisationResolution	The absolute length between successive samples to use when linearising GraphicsPaths.

Definition at line 1431 of file Animation.cs.

7.5.3 Member Function Documentation

7.5.3.1 AddFrame()

```
void VectSharp.Animation.AddFrame (
            Frame frame,
            Transition transition = null )
```

Adds a new frame to the animation, with the specified transition.

Parameters

frame	The new frame to add to the animation.
transition	The transition that should be applied between the previous frame and the new frame. This parameter is ignored for the first frame. If this is `null`, the animation will abruptly change from one frame to the next.

Definition at line 1556 of file Animation.cs.

7.5.3.2 GetFrameAtAbsolute()

```
Page VectSharp.Animation.GetFrameAtAbsolute (
            double time )
```

Obtains the (interpolated) frame that should be displayed after the specified time has passed since the start of the animation.

Parameters

time	The time since the start of the animation (in milliseconds).

Returns

A Page containing the interpolated frame that should be displayed after the specified time has passed since the start of the animation.

Definition at line 1443 of file Animation.cs.

7.5.3.3 GetFrameAtRelative()

```
Page VectSharp.Animation.GetFrameAtRelative (
            double relativeTime )
```

Obtains the (interpolated) frame that should be displayed after the specified relative time has passed since the start of the animation.

Parameters

relativeTime	The time since the start of the animation (ranging from 0 for the start of the animation, to 1 for the end of the animation).

Returns

A Page containing the interpolated frame that should be displayed after the specified time has passed since the start of the animation.

Definition at line 1546 of file Animation.cs.

7.5.3.4 RemoveLastFrame()

```
void VectSharp.Animation.RemoveLastFrame ( )
```

Removes the last frame from the animation (and the corresponding transition).

Exceptions

ArgumentOutOfRangeException	Thrown if there are no frames in the animation.

Definition at line 1578 of file Animation.cs.

7.5.4 Property Documentation

7.5.4.1 Background

```
Colour VectSharp.Animation.Background = Colour.FromRgba(255, 255, 255, 0)  [get], [set]
```

The background colour of the animation.

Definition at line 1408 of file Animation.cs.

7.5.4.2 Duration

```
double VectSharp.Animation.Duration = 0  [get]
```

The total duration of the animation (not including the number of repeats).

Definition at line 1418 of file Animation.cs.

7.5.4.3 Frames

```
ImmutableList<Frame> VectSharp.Animation.Frames = ImmutableList<Frame>.Empty  [get]
```

The key frames of the animation.

Definition at line 1385 of file Animation.cs.

7.5.4.4 Height

```
double VectSharp.Animation.Height  [get], [set]
```

The height of the animation.

Definition at line 1403 of file Animation.cs.

7.5.4.5 LinearisationResolution

```
double VectSharp.Animation.LinearisationResolution  [get]
```

The absolute length between successive samples to use when linearising GraphicsPaths.

Definition at line 1413 of file Animation.cs.

7.5.4.6 RepeatCount

```
int VectSharp.Animation.RepeatCount = 0  [get], [set]
```

The number of times that the animation should repeat.

Definition at line 1423 of file Animation.cs.

7.5.4.7 Transitions

```
ImmutableList<Transition> VectSharp.Animation.Transitions = ImmutableList<Transition>.Empty
[get]
```

The transitions between successive frames of the animation. This array always contains one fewer element than Frames.

Definition at line 1390 of file Animation.cs.

7.5.4.8 Width

```
double VectSharp.Animation.Width  [get], [set]
```

The width of the animation.

Definition at line 1398 of file Animation.cs.

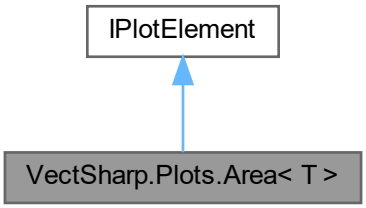
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.6 VectSharp.Plots.Area< T > Class Template Reference

A plot element that fills an area between a line passing through some data points and a base line.

Inheritance diagram for VectSharp.Plots.Area< T >:



Public Member Functions

- Area (IEnumerable< T > data, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)
  Create a new Area< T > instance.
- void Plot (Graphics target)
  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points through which the upper part of the area will pass.
- Func< T, T > GetBaseline `[get, set]`

  A function returning the baseline for each data point.
- bool Smooth `[get, set]`

  If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the area.
- string Tag `[get, set]`

  A tag to identify the area in the plot.

7.6.1 Detailed Description

A plot element that fills an area between a line passing through some data points and a base line.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 599 of file DataPoints.cs.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Area()

```
VectSharp.Plots.Area< T >.Area (
            IEnumerable< T > data,
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Area<T> instance.

Parameters

data	The data points through which the upper part of the area will pass.
getBaseline	A function returning the baseline for each data point.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 640 of file DataPoints.cs.

7.6.3 Member Function Documentation

7.6.3.1 Plot()

```
void VectSharp.Plots.Area< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 648 of file DataPoints.cs.

7.6.4 Property Documentation

7.6.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.Area< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 621 of file DataPoints.cs.

7.6.4.2 Data

```
IEnumerable<T> VectSharp.Plots.Area< T >.Data  [get], [set]
```

The data points through which the upper part of the area will pass.

Definition at line 604 of file DataPoints.cs.

7.6.4.3 GetBaseline

```
Func<T, T> VectSharp.Plots.Area< T >.GetBaseline  [get], [set]
```

A function returning the baseline for each data point.

Definition at line 609 of file DataPoints.cs.

7.6.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Area< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the area.

Definition at line 627 of file DataPoints.cs.

7.6.4.5 Smooth

```
bool VectSharp.Plots.Area< T >.Smooth  [get], [set]
```

If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.

Definition at line 615 of file DataPoints.cs.

7.6.4.6 Tag

```
string VectSharp.Plots.Area< T >.Tag  [get], [set]
```

A tag to identify the area in the plot.

Definition at line 632 of file DataPoints.cs.

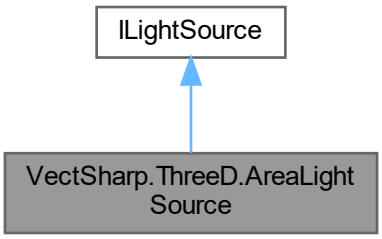
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.7 VectSharp.ThreeD.AreaLightSource Class Reference

Represents a light source emitting light from a circular area.

Inheritance diagram for VectSharp.ThreeD.AreaLightSource:



Public Member Functions

- AreaLightSource (double intensity, Point3D center, double radius, double penumbraRadius, Normalized↩ Vector3D direction, double sourceDistance, int shadowSamplingPointCount)

  Creates a new AreaLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Center `[get]`

The centre of the light-emitting area.

- NormalizedVector3D Direction `[get]`

  The direction of the light's main axis, i.e. the normal to the plane containing the light-emitting area.

- double Radius `[get]`

  The radius of the light emitting area.

- double PenumbraRadius `[get]`

  The radius of the penumbra area.

- double Intensity `[get, set]`

  The base intensity of the light.

- double SourceDistance `[get]`

  The distance between the focal point of the light and the light's Center.

- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

- double PenumbraAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates between the light-emitting area radius and the penumbra radius.

- int ShadowSamplingPointCount `[get]`

  The number of points to use when determining the amount of light that is obstructed at a certain point.

7.7.1 Detailed Description

Represents a light source emitting light from a circular area.

Definition at line 579 of file Lights.cs.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 AreaLightSource()

```
VectSharp.ThreeD.AreaLightSource.AreaLightSource (
            double intensity,
            Point3D center,
            double radius,
            double penumbraRadius,
            NormalizedVector3D direction,
            double sourceDistance,
            int shadowSamplingPointCount )
```

Creates a new AreaLightSource instance.

Parameters

intensity	The base intensity of the light.
center	The centre of the light-emitting area.
radius	The radius of the light-emitting area.
penumbraRadius	The radius of the penumbra area.
direction	The direction of the light.
sourceDistance	The distance between the focal point of the light and the light's center.
shadowSamplingPointCount	The number of points to use when determining the amount of light that is obstructed at a certain point.

Definition at line 643 of file Lights.cs.

7.7.3 Member Function Documentation

7.7.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.AreaLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 690 of file Lights.cs.

7.7.3.2 GetObstruction()

```
double VectSharp.ThreeD.AreaLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 763 of file Lights.cs.

7.7.4 Property Documentation

7.7.4.1 CastsShadow

```
bool VectSharp.ThreeD.AreaLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 582 of file Lights.cs.

7.7.4.2 Center

```
Point3D VectSharp.ThreeD.AreaLightSource.Center  [get]
```

The centre of the light-emitting area.

Definition at line 587 of file Lights.cs.

7.7.4.3 Direction

```
NormalizedVector3D VectSharp.ThreeD.AreaLightSource.Direction  [get]
```

The direction of the light's main axis, i.e. the normal to the plane containing the light-emitting area.

Definition at line 594 of file Lights.cs.

7.7.4.4 DistanceAttenuationExponent

```
double VectSharp.ThreeD.AreaLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 619 of file Lights.cs.

7.7.4.5 Intensity

```
double VectSharp.ThreeD.AreaLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 609 of file Lights.cs.

7.7.4.6 PenumbraAttenuationExponent

```
double VectSharp.ThreeD.AreaLightSource.PenumbraAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates between the light-emitting area radius and the penumbra radius.

Definition at line 624 of file Lights.cs.

7.7.4.7 PenumbraRadius

```
double VectSharp.ThreeD.AreaLightSource.PenumbraRadius  [get]
```

The radius of the penumbra area.

Definition at line 604 of file Lights.cs.

7.7.4.8 Radius

```
double VectSharp.ThreeD.AreaLightSource.Radius  [get]
```

The radius of the light emitting area.

Definition at line 599 of file Lights.cs.

7.7.4.9 ShadowSamplingPointCount

```
int VectSharp.ThreeD.AreaLightSource.ShadowSamplingPointCount  [get]
```

The number of points to use when determining the amount of light that is obstructed at a certain point.

Definition at line 629 of file Lights.cs.

7.7.4.10 SourceDistance

```
double VectSharp.ThreeD.AreaLightSource.SourceDistance  [get]
```

The distance between the focal point of the light and the light's Center.

Definition at line 614 of file Lights.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.8 VectSharp.Canvas.AvaloniaContextInterpreter Class Reference

Contains methods to render a Page to an Avalonia.Controls.Canvas.

Public Types

- enum TextOptions

  Defines whether text items should be converted into paths when drawing.

Static Public Member Functions

- static Avalonia.Controls.Canvas PaintToCanvas (this Page page, TextOptions textOption=TextOptions.↩ ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas.
- static Avalonia.Controls.Control PaintToCanvas (this Page page, bool graphicsAsControls, TextOptions text↩ Option=TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static Avalonia.Controls.Control PaintToCanvas (this Page page, bool graphicsAsControls, Dictionary< string, Delegate > taggedActions, bool removeTaggedActionsAfterExecution=true, TextOptions textOption=Text↩ Options.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static Avalonia.Controls.Canvas PaintToCanvas (this Page page, Dictionary< string, Delegate > tagged↩ Actions, bool removeTaggedActionsAfterExecution=true, TextOptions textOption=TextOptions.ConvertIf↩ Necessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static AnimatedCanvas PaintToAnimatedCanvas (this Animation animation, double frameRate=60, dou- ble durationScaling=1, TextOptions textOption=TextOptions.ConvertIfNecessary, FilterOption filter↩ Option=default)

  Render an Animation to an Avalonia.Controls.Control.

7.8.1 Detailed Description

Contains methods to render a Page to an Avalonia.Controls.Canvas.

Definition at line 2605 of file AvaloniaContext.cs.

7.8.2 Member Enumeration Documentation

7.8.2.1 TextOptions

```
enum VectSharp.Canvas.AvaloniaContextInterpreter.TextOptions
```

Defines whether text items should be converted into paths when drawing.

Definition at line 2610 of file AvaloniaContext.cs.

7.8.3 Member Function Documentation

7.8.3.1 PaintToAnimatedCanvas()

```
static AnimatedCanvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToAnimatedCanvas (
            this Animation animation,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render an Animation to an Avalonia.Controls.Control.

Parameters

animation	The Animation to render.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered animation.

Definition at line 2728 of file AvaloniaContext.cs.

7.8.3.2 PaintToCanvas() [1/4]

```
static Avalonia.Controls.Control VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
```

```
            bool graphicsAsControls,
            Dictionary< string, Delegate > taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
graphicsAsControls	If this is true, each graphics object (e.g. paths, text...) is rendered as a separate Avalonia.Controls.Control. Otherwise, they are directly rendered onto the drawing context (which is faster, but does not allow interactivity).
taggedActions	A Dictionary<String, Delegate> containing the Actions that will be performed on items with the corresponding tag. If *graphicsAsControls* is true, the delegates should be voids that accept one parameter of type TextBlock or Path (depending on the tagged item), otherwise, they should accept one parameter of type RenderAction and return an IEnumerable<RenderAction> of the actions that will actually be performed.
removeTaggedActionsAfterExecution	Whether the Actions should be removed from *taggedActions* after their execution. Set to false if the same Action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2680 of file AvaloniaContext.cs.

7.8.3.3 PaintToCanvas() [2/4]

```
static Avalonia.Controls.Control VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            bool graphicsAsControls,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
graphicsAsControls	If this is true, each graphics object (e.g. paths, text...) is rendered as a separate Avalonia.Controls.Control. Otherwise, they are directly rendered onto the drawing context (which is faster, but does not allow interactivity).
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2653 of file AvaloniaContext.cs.

7.8.3.4 PaintToCanvas() [3/4]

```
static Avalonia.Controls.Canvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            Dictionary< string, Delegate > taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
taggedActions	A Dictionary<String, Delegate> containing the Actions that will be performed on items with the corresponding tag. The delegates should accept one parameter of type TextBlock or Path (depending on the tagged item).
removeTaggedActionsAfterExecution	Whether the Actions should be removed from *taggedActions* after their execution. Set to false if the same Action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2706 of file AvaloniaContext.cs.

7.8.3.5 PaintToCanvas() [4/4]

```
static Avalonia.Controls.Canvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 2635 of file AvaloniaContext.cs.

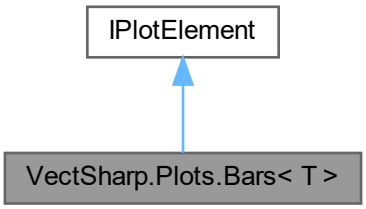
The documentation for this class was generated from the following file:

- VectSharp.Canvas/AvaloniaContext.cs

7.9 VectSharp.Plots.Bars< T > Class Template Reference

A plot element that draws bars.

Inheritance diagram for VectSharp.Plots.Bars< T >:



Public Member Functions

- Bars (IEnumerable< T > data, IComparer< T > sorting, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- Bars (IEnumerable< T > data, Comparison< T > sorting, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- Bars (IReadOnlyList< T > data, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

- Bars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IRead↩ OnlyList< double > > coordinateSystem)

  Create a new Bars instance.
- Bars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IRead↩ OnlyList< double > > coordinateSystem)

  Create a new Bars instance.
- Bars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new Bars instance.

Properties

- SortedSet< T > Data `[get, set]`

  The data points corresponding to the tips of the bars.
- Func< T, T > GetBaseline `[get, set]`

  A function that returns the bottom for each bar. This function should accept a single parameter of type T and return another T object, representing the bottom of the bar in data space.
- double Margin `[get, set]`

  The margin between consecutive bars. This should range between 0 and 1.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes for the bars.
- string Tag `[get, set]`

  A tag to identify the bars in the plot.

7.9.1 Detailed Description

A plot element that draws bars.

A plot element that draws bars for numerical data.

Template Parameters

T	The type of data elements.

Definition at line 29 of file Bars.cs.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Bars() [1/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< T > data,
            IComparer< T > sorting,
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 89 of file Bars.cs.

7.9.2.2 Bars() [2/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< T > data,
            Comparison< T > sorting,
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 105 of file Bars.cs.

7.9.2.3 Bars() [3/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IReadOnlyList< T > data,
```

```
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars. These should already be sorted.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 115 of file Bars.cs.

7.9.2.4 Bars() [4/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 299 of file Bars.cs.

7.9.2.5 Bars() [5/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 310 of file Bars.cs.

7.9.2.6 Bars() [6/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 319 of file Bars.cs.

7.9.3 Member Function Documentation

7.9.3.1 Plot()

```
void VectSharp.Plots.Bars< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 143 of file Bars.cs.

7.9.4 Property Documentation

7.9.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.Bars< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 67 of file Bars.cs.

7.9.4.2 Data

```
SortedSet<T> VectSharp.Plots.Bars< T >.Data  [get], [set]
```

The data points corresponding to the tips of the bars.

Definition at line 36 of file Bars.cs.

7.9.4.3 GetBaseline

```
Func<T, T> VectSharp.Plots.Bars< T >.GetBaseline  [get], [set]
```

A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.

Definition at line 43 of file Bars.cs.

7.9.4.4 Margin

```
double VectSharp.Plots.Bars< T >.Margin  [get], [set]
```

The margin between consecutive bars. This should range between 0 and 1.

Definition at line 48 of file Bars.cs.

7.9.4.5 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Bars< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes for the bars.

Definition at line 73 of file Bars.cs.

7.9.4.6 Tag

```
string VectSharp.Plots.Bars< T >.Tag  [get], [set]
```

A tag to identify the bars in the plot.

Definition at line 78 of file Bars.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.10 VectSharp.TrueTypeFile.Bearings Struct Reference

Represents the left- and right-side bearings of a glyph.

Public Attributes

- int LeftSideBearing

  The left-side bearing of the glyph.
- int RightSideBearing

  The right-side bearing of the glyph.

7.10.1 Detailed Description

Represents the left- and right-side bearings of a glyph.

Definition at line 2319 of file TrueType.cs.

7.10.2 Member Data Documentation

7.10.2.1 LeftSideBearing

```
int VectSharp.TrueTypeFile.Bearings.LeftSideBearing
```

The left-side bearing of the glyph.

Definition at line 2324 of file TrueType.cs.

7.10.2.2 RightSideBearing

```
int VectSharp.TrueTypeFile.Bearings.RightSideBearing
```

The right-side bearing of the glyph.

Definition at line 2329 of file TrueType.cs.

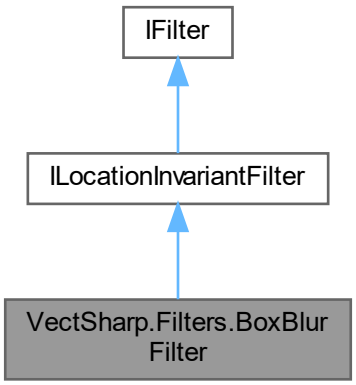
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.11 VectSharp.Filters.BoxBlurFilter Class Reference

Represents a filter applying a box blur.

Inheritance diagram for VectSharp.Filters.BoxBlurFilter:



Public Member Functions

- BoxBlurFilter (double boxRadius)

  Creates a new BoxBlurFilter with the specified radius.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- double BoxRadius `[get]`

  The radius of the box blur (the actual size of the box is 2 ∗ BoxRadius + 1).
- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.11.1 Detailed Description

Represents a filter applying a box blur.

Definition at line 26 of file BoxBlurFilter.cs.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 BoxBlurFilter()

```
VectSharp.Filters.BoxBlurFilter.BoxBlurFilter (
            double boxRadius )
```

Creates a new BoxBlurFilter with the specified radius.

Parameters

boxRadius	The radius of the box blur (the actual size of the box is 2 ∗ *boxRadius* + 1).

Definition at line 42 of file BoxBlurFilter.cs.

7.11.3 Member Function Documentation

7.11.3.1 Filter()

```
RasterImage VectSharp.Filters.BoxBlurFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 50 of file BoxBlurFilter.cs.

7.11.4 Property Documentation

7.11.4.1 BottomRightMargin

```
Point VectSharp.Filters.BoxBlurFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 36 of file BoxBlurFilter.cs.

7.11.4.2 BoxRadius

```
double VectSharp.Filters.BoxBlurFilter.BoxRadius  [get]
```

The radius of the box blur (the actual size of the box is 2 ∗ BoxRadius + 1).

Definition at line 31 of file BoxBlurFilter.cs.

7.11.4.3 TopLeftMargin

```
Point VectSharp.Filters.BoxBlurFilter.TopLeftMargin [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 34 of file BoxBlurFilter.cs.

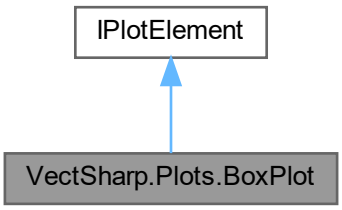
The documentation for this class was generated from the following file:

- VectSharp/Filters/BoxBlurFilter.cs

7.12 VectSharp.Plots.BoxPlot Class Reference

A plot element that draws a box plot.

Inheritance diagram for VectSharp.Plots.BoxPlot:



Public Member Functions

- BoxPlot (IReadOnlyList< double > position, IReadOnlyList< double > direction, double whisker1, double box1, double box2, double whisker2, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new BoxPlot instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Position [get, set]

  The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
- IReadOnlyList< double > Direction [get, set]

  The direction along which the box plot is drawn, in data space coordinates.
- double Whisker1 [get, set]

  The distance between the centre of the box plot and the first whisker, expressed as a multiple of Direction.
- double Box1 [get, set]

  The distance between the centre of the box plot and the first side of the box, expressed as a multiple of Direction.
- double Box2 [get, set]

  The distance between the centre of the box plot and the second side of the box, expressed as a multiple of Direction.
- double Whisker2 [get, set]

  The distance between the centre of the box plot and the second whisker, expressed as a multiple of Direction.
- double Width = 10 [get, set]

  The width of the box plot, in data space coordinates.
- double WhiskerWidth = 0.5 [get, set]

  The width of the whiskers, expressed as a multiple of the Width of the box.
- double NotchWidth = 0.5 [get, set]

  The width of the notch, expressed as a multiple of the Width of the box.
- double NotchSize = 0 [get, set]

  The distance between the centre of the box plot and the end of the notch, expressed as a multiple of Direction.
- PlotElementPresentationAttributes BoxPresentationAttributes = new PlotElementPresentationAttributes() { Fill = Colours.White } [get, set]

  Presentation attributes for the box.
- PlotElementPresentationAttributes WhiskersPresentationAttributes = new PlotElementPresentationAttributes() { Fill = null } [get, set]

  Presentation attributes for the whiskers.
- PlotElementPresentationAttributes CentreSymbolPresentationAttributes = new PlotElementPresentationAttributes() { Fill = null, Stroke = null } [get, set]

  Presentation attributes for the symbol drawn at the centre of the box plot.
- IDataPointElement CentreSymbol = new PathDataPointElement() [get, set]

  Symbol drawn at the centre of the box plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- string Tag [get, set]

  A tag to identify the box in the plot.

7.12.1 Detailed Description

A plot element that draws a box plot.

Definition at line 26 of file BoxPlot.cs.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 BoxPlot()

```
VectSharp.Plots.BoxPlot.BoxPlot (
            IReadOnlyList< double > position,
            IReadOnlyList< double > direction,
            double whisker1,
            double box1,
            double box2,
            double whisker2,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new BoxPlot instance.

Parameters

position	The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
direction	The direction along which the box plot is drawn, in data space coordinates.
whisker1	The distance between the centre of the box plot and the first whisker, expressed as a multiple of *direction* .
box1	The distance between the centre of the box plot and the first side of the box, expressed as a multiple of *direction* .
box2	The distance between the centre of the box plot and the second side of the box, expressed as a multiple of *direction* .
whisker2	The distance between the centre of the box plot and the second whisker, expressed as a multiple of *direction* .
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 129 of file BoxPlot.cs.

7.12.3 Member Function Documentation

7.12.3.1 Plot()

```
void VectSharp.Plots.BoxPlot.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 141 of file BoxPlot.cs.

7.12.4 Property Documentation

7.12.4.1 Box1

```
double VectSharp.Plots.BoxPlot.Box1  [get], [set]
```

The distance between the centre of the box plot and the first side of the box, expressed as a multiple of Direction.

Definition at line 48 of file BoxPlot.cs.

7.12.4.2 Box2

```
double VectSharp.Plots.BoxPlot.Box2  [get], [set]
```

The distance between the centre of the box plot and the second side of the box, expressed as a multiple of Direction.

Definition at line 54 of file BoxPlot.cs.

7.12.4.3 BoxPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.BoxPresentationAttributes = new
PlotElementPresentationAttributes() { Fill = Colours.White }  [get], [set]
```

Presentation attributes for the box.

Definition at line 86 of file BoxPlot.cs.

7.12.4.4 CentreSymbol

```
IDataPointElement VectSharp.Plots.BoxPlot.CentreSymbol = new PathDataPointElement()  [get],
[set]
```

Symbol drawn at the centre of the box plot.

Definition at line 101 of file BoxPlot.cs.

7.12.4.5 CentreSymbolPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.CentreSymbolPresentationAttributes =
new PlotElementPresentationAttributes() { Fill = null, Stroke = null }  [get], [set]
```

Presentation attributes for the symbol drawn at the centre of the box plot.

Definition at line 96 of file BoxPlot.cs.

7.12.4.6 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.BoxPlot.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 106 of file BoxPlot.cs.

7.12.4.7 Direction

```
IReadOnlyList<double> VectSharp.Plots.BoxPlot.Direction  [get], [set]
```

The direction along which the box plot is drawn, in data space coordinates.

Definition at line 36 of file BoxPlot.cs.

7.12.4.8 NotchSize

```
double VectSharp.Plots.BoxPlot.NotchSize = 0  [get], [set]
```

The distance between the centre of the box plot and the end of the notch, expressed as a multiple of Direction.

Definition at line 81 of file BoxPlot.cs.

7.12.4.9 NotchWidth

```
double VectSharp.Plots.BoxPlot.NotchWidth = 0.5  [get], [set]
```

The width of the notch, expressed as a multiple of the Width of the box.

Definition at line 75 of file BoxPlot.cs.

7.12.4.10 Position

```
IReadOnlyList<double> VectSharp.Plots.BoxPlot.Position  [get], [set]
```

The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.

Definition at line 31 of file BoxPlot.cs.

7.12.4.11 Tag

```
string VectSharp.Plots.BoxPlot.Tag  [get], [set]
```

A tag to identify the box in the plot.

Definition at line 111 of file BoxPlot.cs.

7.12.4.12 Whisker1

```
double VectSharp.Plots.BoxPlot.Whisker1  [get], [set]
```

The distance between the centre of the box plot and the first whisker, expressed as a multiple of Direction.

Definition at line 42 of file BoxPlot.cs.

7.12.4.13 Whisker2

```
double VectSharp.Plots.BoxPlot.Whisker2  [get], [set]
```

The distance between the centre of the box plot and the second whisker, expressed as a multiple of Direction.

Definition at line 60 of file BoxPlot.cs.

7.12.4.14 WhiskersPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.WhiskersPresentationAttributes = new
PlotElementPresentationAttributes() { Fill = null }  [get], [set]
```

Presentation attributes for the whiskers.

Definition at line 91 of file BoxPlot.cs.

7.12.4.15 WhiskerWidth

```
double VectSharp.Plots.BoxPlot.WhiskerWidth = 0.5  [get], [set]
```

The width of the whiskers, expressed as a multiple of the Width of the box.

Definition at line 70 of file BoxPlot.cs.

7.12.4.16 Width

```
double VectSharp.Plots.BoxPlot.Width = 10  [get], [set]
```

The width of the box plot, in data space coordinates.

Definition at line 65 of file BoxPlot.cs.

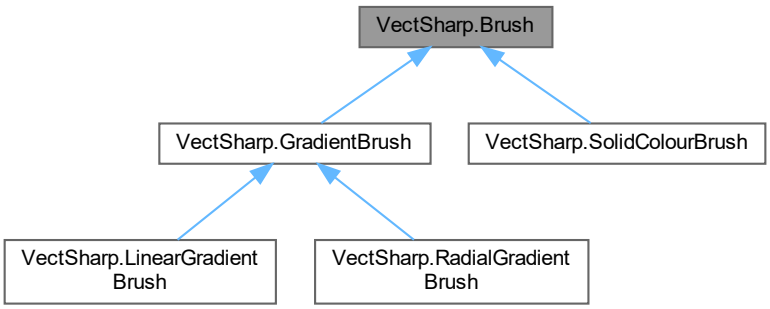
The documentation for this class was generated from the following file:

- VectSharp.Plots/BoxPlot.cs

7.13 VectSharp.Brush Class Reference

Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.

Inheritance diagram for VectSharp.Brush:



Public Member Functions

- abstract Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Static Public Member Functions

- static implicit operator Brush (Colour colour)

  Implicitly converts a Colour into a SolidColourBrush.

7.13.1 Detailed Description

Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.

Definition at line 30 of file Brush.cs.

7.13.2 Member Function Documentation

7.13.2.1 MultiplyOpacity()

```
abstract Brush VectSharp.Brush.MultiplyOpacity (
            double opacity )  [pure virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implemented in VectSharp.SolidColourBrush, VectSharp.LinearGradientBrush, and VectSharp.RadialGradientBrush.

7.13.2.2 operator Brush()

```
static implicit VectSharp.Brush.operator Brush (
            Colour colour )  [static]
```

Implicitly converts a Colour into a SolidColourBrush.

Parameters

colour	The Colour to use for the brush.

Definition at line 45 of file Brush.cs.

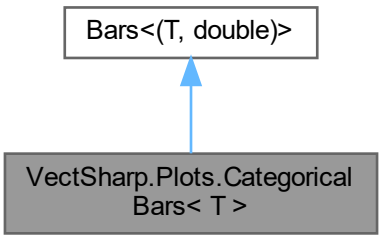
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.14 VectSharp.Plots.CategoricalBars< T > Class Template Reference

A plot element that draws bars for categorical data.

Inheritance diagram for VectSharp.Plots.CategoricalBars< T >:



Public Member Functions

- CategoricalBars (IEnumerable<(T, double)> data, IComparer<(T, double)> sorting, Func<(T, double),(T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new CategoricalBars<T> instance.
- CategoricalBars (IEnumerable<(T, double)> data, Comparison<(T, double)> sorting, Func<(T, double),(T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new CategoricalBars<T> instance.
- CategoricalBars (IEnumerable<(T, double)> data, Comparison<(T, double)> sorting, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.
- CategoricalBars (IReadOnlyList<(T, double)> data, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Additional Inherited Members

7.14.1 Detailed Description

A plot element that draws bars for categorical data.

Template Parameters

T	The type of the data categories.

Definition at line 245 of file Bars.cs.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 CategoricalBars() [1/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            IComparer<(T, double)> sorting,
            Func<(T, double),(T, double)> getBaseline,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new CategoricalBars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 256 of file Bars.cs.

7.14.2.2 CategoricalBars() [2/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            Comparison<(T, double)> sorting,
            Func<(T, double),(T, double)> getBaseline,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new CategoricalBars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 267 of file Bars.cs.

7.14.2.3 CategoricalBars() [3/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            Comparison<(T, double)> sorting,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 275 of file Bars.cs.

7.14.2.4 CategoricalBars() [4/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IReadOnlyList<(T, double)> data,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Parameters

data	The data points corresponding to the tips of the bars. These should already be sorted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 282 of file Bars.cs.

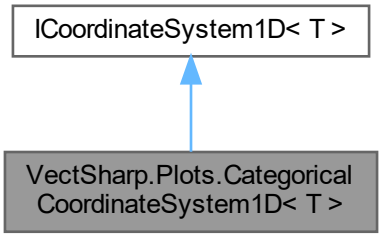
The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.15 VectSharp.Plots.CategoricalCoordinateSystem1D< T > Class Template Reference

Represents a categorical 1-D coordinate system.

Inheritance diagram for VectSharp.Plots.CategoricalCoordinateSystem1D< T >:



Public Member Functions

- CategoricalCoordinateSystem1D (Dictionary< T, double > coordinates)

  Creates a new CategoricalCoordinateSystem1D<T> using the coordinates specified in the supplied coordinates Dictionary<TKey, TValue>.
- CategoricalCoordinateSystem1D (IReadOnlyList< T > data, double scale=350)

  Creates a new CategoricalCoordinateSystem1D<T> assigning equally-spaced coordinates to all different values of data .
- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`*.*

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A `double` *representing the dataPoint in plot space.*

Properties

- Dictionary< T, double > Coordinates `[get, set]`

  A Dictionary<TKey, TValue> storing the coordinates.

7.15.1 Detailed Description

Represents a categorical 1-D coordinate system.

Template Parameters

T	The type of data elements.

Definition at line 1082 of file CoordinateSystems.cs.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 CategoricalCoordinateSystem1D() [1/2]

```
VectSharp.Plots.CategoricalCoordinateSystem1D< T >.CategoricalCoordinateSystem1D (
            Dictionary< T, double > coordinates )
```

Creates a new CategoricalCoordinateSystem1D<T> using the coordinates specified in the supplied *coordinates* Dictionary<TKey, TValue>.

Parameters

coordinates	

Definition at line 1093 of file CoordinateSystems.cs.

7.15.2.2 CategoricalCoordinateSystem1D() [2/2]

```
VectSharp.Plots.CategoricalCoordinateSystem1D< T >.CategoricalCoordinateSystem1D (
            IReadOnlyList< T > data,
            double scale = 350 )
```

Creates a new CategoricalCoordinateSystem1D<T> assigning equally-spaced coordinates to all different values of *data* .

Parameters

data	The data containing discrete parameter values that should be assigned to different coordinates.
scale	The maximum coordinate.

Definition at line 1103 of file CoordinateSystems.cs.

7.15.3 Member Function Documentation

7.15.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.CategoricalCoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem1D< T >.

Definition at line 1125 of file CoordinateSystems.cs.

7.15.4 Property Documentation

7.15.4.1 Coordinates

```
Dictionary<T, double> VectSharp.Plots.CategoricalCoordinateSystem1D< T >.Coordinates  [get],
[set]
```

A Dictionary<TKey, TValue> storing the coordinates.

Definition at line 1087 of file CoordinateSystems.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.16 VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord Struct Reference

Public Member Functions

- ClassRangeRecord (Stream fs)

Public Attributes

- ushort StartGlyphID
- ushort EndGlyphID
- ushort Class

7.16.1 Detailed Description

Definition at line 3925 of file TrueType.cs.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 ClassRangeRecord()

```
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.ClassRangeRecord (
            Stream fs )
```

Definition at line 3931 of file TrueType.cs.

7.16.3 Member Data Documentation

7.16.3.1 Class

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.Class
```

Definition at line 3929 of file TrueType.cs.

7.16.3.2 EndGlyphID

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.EndGlyphID
```

Definition at line 3928 of file TrueType.cs.

7.16.3.3 StartGlyphID

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.StartGlyphID
```

Definition at line 3927 of file TrueType.cs.

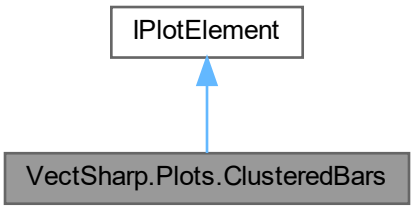
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.17 VectSharp.Plots.ClusteredBars Class Reference

A plot element that draws clusters of bars.

Inheritance diagram for VectSharp.Plots.ClusteredBars:



Public Member Functions

- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new ClusteredBars instance.
- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new ClusteredBars instance.
- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new ClusteredBars instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- bool Vertical = true `[get, set]`

  If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.
- SortedSet< IReadOnlyList< double > > Data `[get, set]`

  The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
- Func< IReadOnlyList< double >, IReadOnlyList< double > > GetBaseline `[get, set]`

A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of doubles), and return another object of the same type, representing the bottom of the cluster in data space.

- double InterClusterMargin [get, set]

  The margin between consecutive bar clusters.
- double IntraClusterMargin [get, set]

  The margin between consecutive bars within a single cluster.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } [get, set]

  Presentation attributes for the bars. An element from this collection is used for each bar in the cluster; if there are more bars than elements in this collection, the presentation attributes are wrapped.
- string Tag [get, set]

  A tag to identify the clustered bars in the plot.

7.17.1 Detailed Description

A plot element that draws clusters of bars.

Definition at line 595 of file Bars.cs.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 ClusteredBars() [1/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is true, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
sorting	A comparer used to sort the bar clusters.
getBaseline	A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of doubles), and return another object of the same type, representing the bottom of the cluster in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 691 of file Bars.cs.

7.17.2.2 ClusteredBars() [2/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
sorting	A comparer used to sort the bar clusters.
getBaseline	A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the cluster in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 710 of file Bars.cs.

7.17.2.3 ClusteredBars() [3/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 722 of file Bars.cs.

7.17.3 Member Function Documentation

7.17.3.1 Plot()

```
void VectSharp.Plots.ClusteredBars.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 725 of file Bars.cs.

7.17.4 Property Documentation

7.17.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.CoordinateSystem
[get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 663 of file Bars.cs.

7.17.4.2 Data

```
SortedSet<IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.Data  [get], [set]
```

The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.

Definition at line 613 of file Bars.cs.

7.17.4.3 GetBaseline

```
Func<IReadOnlyList<double>, IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.Get↩
Baseline  [get], [set]
```

A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IRead↩ OnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the cluster in data space.

Definition at line 620 of file Bars.cs.

7.17.4.4 InterClusterMargin

```
double VectSharp.Plots.ClusteredBars.InterClusterMargin  [get], [set]
```

The margin between consecutive bar clusters.

Definition at line 625 of file Bars.cs.

7.17.4.5 IntraClusterMargin

```
double VectSharp.Plots.ClusteredBars.IntraClusterMargin  [get], [set]
```

The margin between consecutive bars within a single cluster.

Definition at line 644 of file Bars.cs.

7.17.4.6 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.ClusteredBars.Presentation↩
Attributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes()
}  [get], [set]
```

Presentation attributes for the bars. An element from this collection is used for each bar in the cluster; if there are more bars than elements in this collection, the presentation attributes are wrapped.

Definition at line 671 of file Bars.cs.

7.17.4.7 Tag

```
string VectSharp.Plots.ClusteredBars.Tag  [get], [set]
```

A tag to identify the clustered bars in the plot.

Definition at line 676 of file Bars.cs.

7.17.4.8 Vertical

```
bool VectSharp.Plots.ClusteredBars.Vertical = true  [get], [set]
```

If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 604 of file Bars.cs.

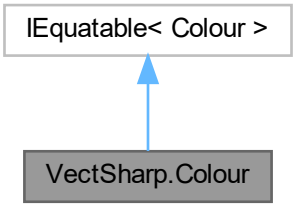
The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.18 VectSharp.Colour Struct Reference

Represents an RGB colour.

Inheritance diagram for VectSharp.Colour:



Public Member Functions

- override bool Equals (object obj)
- bool Equals (Colour col)
- override int GetHashCode ()
- string ToCSSString (bool includeAlpha)

  Convert the Colour object into a hex string that is constituted by a "#" followed by two-digit hexadecimal representations of the red, green and blue components of the colour (in the range 0x00 - 0xFF). Optionally also includes opacity (alpha channel) data.
- Colour WithAlpha (double alpha)

  Create a new Colour with the same RGB components as the current Colour, but with the specified alpha .
- Colour WithAlpha (byte alpha)

  Create a new Colour with the same RGB components as the current Colour, but with the specified alpha .
- double double double Z ToXYZ ()
- double double double b ToLab ()
- double double double L ToHSL ()

Static Public Member Functions

- static Colour FromRgb (double r, double g, double b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgb (byte r, byte g, byte b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgb (int r, int g, int b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgba (double r, double g, double b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (byte r, byte g, byte b, byte a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (byte r, byte g, byte b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (int r, int g, int b, int a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (int r, int g, int b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba ((int r, int g, int b, double a) colour)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static bool operator== (Colour col1, Colour col2)
- static bool operator!= (Colour col1, Colour col2)
- static ? Colour FromCSSString (string cssString)

  Convert a CSS colour string into a Colour object.
- static Colour WithAlpha (Colour original, double alpha)

  Create a new Colour with the same RGB components as the original Colour, but with the specified alpha .
- static Colour WithAlpha (Colour original, byte alpha)

  Create a new Colour with the same RGB components as the original Colour, but with the specified alpha .
- static Colour FromXYZ (double x, double y, double z)

  Creates a Colour from CIE XYZ coordinates.
- static Colour FromLab (double L, double a, double b)

  Creates a Colour from CIE Lab coordinates (under Illuminant D65).
- static Colour FromHSL (double h, double s, double l)

  Creates a Colour from HSL coordinates.

Public Attributes

- double R

  Red component of the colour. Range: [0, 1].
- double G

  Green component of the colour. Range: [0, 1].
- double B

  Blue component of the colour. Range: [0, 1].
- double A

  Alpha component of the colour. Range: [0, 1].
- double X

  Converts a Colour to the CIE XYZ colour space.
- double double Y
- double L

  Converts a Colour to the CIE Lab colour space (under Illuminant D65).
- double double a
- double H

  Converts a Colour to the HSL colour space.
- double double S

7.18.1 Detailed Description

Represents an RGB colour.

Definition at line 25 of file Colour.cs.

7.18.2 Member Function Documentation

7.18.2.1 Equals() [1/2]

```
bool VectSharp.Colour.Equals (
            Colour col )
```

Definition at line 179 of file Colour.cs.

7.18.2.2 Equals() [2/2]

```
override bool VectSharp.Colour.Equals (
            object obj )
```

Definition at line 166 of file Colour.cs.

7.18.2.3 FromCSSString()

```
static ?  Colour VectSharp.Colour.FromCSSString (
            string cssString )  [static]
```

Convert a CSS colour string into a Colour object.

Parameters

cssString	The CSS colour string. In addition to 148 standard colour names (case-insensitive), #RGB, #RGBA, #RRGGBB and #RRGGBBAA hex strings and rgb(r, g, b) and rgba(r, g, b, a) functional colour notations are supported.

Returns

Definition at line 225 of file Colour.cs.

7.18.2.4 FromHSL()

```
static Colour VectSharp.Colour.FromHSL (
            double h,
            double s,
            double l )  [static]
```

Creates a Colour from HSL coordinates.

Parameters

h	The H component. Should be in range [0, 1].
s	The S component. Should be in range [0, 1].
l	The L component. Should be in range [0, 1].

Returns

A Colour created from the specified components.

Definition at line 582 of file Colour.cs.

7.18.2.5 FromLab()

```
static Colour VectSharp.Colour.FromLab (
            double L,
            double a,
            double b )  [static]
```

Creates a Colour from CIE Lab coordinates (under Illuminant D65).

Parameters

L	The L∗ component.
a	The a∗ component.
b	The b∗ component.

Returns

An sRGB Colour created from the specified components.

Definition at line 504 of file Colour.cs.

7.18.2.6 FromRgb() [1/3]

```
static Colour VectSharp.Colour.FromRgb (
            byte r,
```

```
            byte g,
            byte b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 74 of file Colour.cs.

7.18.2.7 FromRgb() [2/3]

```
static Colour VectSharp.Colour.FromRgb (
            double r,
            double g,
            double b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 1].
g	The green component of the colour. Range: [0, 1].
b	The blue component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 62 of file Colour.cs.

7.18.2.8 FromRgb() [3/3]

```
static Colour VectSharp.Colour.FromRgb (
            int r,
            int g,
            int b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 86 of file Colour.cs.

7.18.2.9 FromRgba() [1/6]

```
static Colour VectSharp.Colour.FromRgba (
            (int r, int g, int b, double a) colour )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

colour	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 160 of file Colour.cs.

7.18.2.10 FromRgba() [2/6]

```
static Colour VectSharp.Colour.FromRgba (
            byte r,
            byte g,
            byte b,
            byte a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 255].

Returns

A ColourColour struct with the specified components.

Definition at line 112 of file Colour.cs.

7.18.2.11 FromRgba() [3/6]

```
static Colour VectSharp.Colour.FromRgba (
            byte r,
            byte g,
            byte b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 125 of file Colour.cs.

7.18.2.12 FromRgba() [4/6]

```
static Colour VectSharp.Colour.FromRgba (
            double r,
            double g,
            double b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 1].
g	The green component of the colour. Range: [0, 1].
b	The blue component of the colour. Range: [0, 1].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 99 of file Colour.cs.

7.18.2.13 FromRgba() [5/6]

```
static Colour VectSharp.Colour.FromRgba (
            int r,
            int g,
            int b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 150 of file Colour.cs.

7.18.2.14 FromRgba() [6/6]

```
static Colour VectSharp.Colour.FromRgba (
            int r,
            int g,
            int b,
            int a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components.

Definition at line 137 of file Colour.cs.

7.18.2.15 FromXYZ()

```
static Colour VectSharp.Colour.FromXYZ (
            double x,
            double y,
            double z )  [static]
```

Creates a Colour from CIE XYZ coordinates.

Parameters

x	The X coordinate.
y	The Y coordinate.
z	The Z coordinate.

Returns

An sRGB Colour created from the specified components.

Definition at line 422 of file Colour.cs.

7.18.2.16 GetHashCode()

```
override int VectSharp.Colour.GetHashCode ( )
```

Definition at line 197 of file Colour.cs.

7.18.2.17 operator"!=()

```
static bool VectSharp.Colour.operator!= (
            Colour col1,
            Colour col2 )  [static]
```

Definition at line 191 of file Colour.cs.

7.18.2.18 operator==()

```
static bool VectSharp.Colour.operator== (
            Colour col1,
            Colour col2 )  [static]
```

Definition at line 185 of file Colour.cs.

7.18.2.19 ToCSSString()

```
string VectSharp.Colour.ToCSSString (
            bool includeAlpha )
```

Convert the Colour object into a hex string that is constituted by a "#" followed by two-digit hexadecimal representations of the red, green and blue components of the colour (in the range 0x00 - 0xFF). Optionally also includes opacity (alpha channel) data.

Parameters

includeAlpha	Whether two additional hex digits representing the colour's opacity (alpha channel) should be included in the string.

Returns

A hex colour string.

Definition at line 208 of file Colour.cs.

7.18.2.20 ToHSL()

```
double double double L VectSharp.Colour.ToHSL ( )
```

Definition at line 535 of file Colour.cs.

7.18.2.21 ToLab()

```
double double double b VectSharp.Colour.ToLab ( )
```

Definition at line 466 of file Colour.cs.

7.18.2.22 ToXYZ()

```
double double double Z VectSharp.Colour.ToXYZ ( )
```

Definition at line 377 of file Colour.cs.

7.18.2.23 WithAlpha() [1/4]

```
Colour VectSharp.Colour.WithAlpha (
            byte alpha )
```

Create a new Colour with the same RGB components as the current Colour, but with the specified *alpha* .

Parameters

alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the current Colour and the specified *alpha* .

Definition at line 368 of file Colour.cs.

7.18.2.24 WithAlpha() [2/4]

```
static Colour VectSharp.Colour.WithAlpha (
            Colour original,
            byte alpha )  [static]
```

Create a new Colour with the same RGB components as the *original* Colour, but with the specified *alpha* .

Parameters

original	The original Colour from which the RGB components will be taken.
alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the *original* Colour and the specified *alpha* .

Definition at line 348 of file Colour.cs.

7.18.2.25 WithAlpha() [3/4]

```
static Colour VectSharp.Colour.WithAlpha (
            Colour original,
            double alpha )  [static]
```

Create a new Colour with the same RGB components as the *original* Colour, but with the specified *alpha* .

Parameters

original	The original Colour from which the RGB components will be taken.
alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the *original* Colour and the specified *alpha* .

Definition at line 337 of file Colour.cs.

7.18.2.26 WithAlpha() [4/4]

```
Colour VectSharp.Colour.WithAlpha (
            double alpha )
```

Create a new Colour with the same RGB components as the current Colour, but with the specified *alpha* .

Parameters

alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the current Colour and the specified *alpha* .

Definition at line 358 of file Colour.cs.

7.18.3 Member Data Documentation

7.18.3.1 A

```
double VectSharp.Colour.A
```

Alpha component of the colour. Range: [0, 1].

Definition at line 45 of file Colour.cs.

7.18.3.2 a

```
double double VectSharp.Colour.a
```

Definition at line 466 of file Colour.cs.

7.18.3.3 B

```
double VectSharp.Colour.B
```

Blue component of the colour. Range: [0, 1].

Definition at line 40 of file Colour.cs.

7.18.3.4 G

```
double VectSharp.Colour.G
```

Green component of the colour. Range: [0, 1].

Definition at line 35 of file Colour.cs.

7.18.3.5 H

```
double VectSharp.Colour.H
```

Converts a Colour to the HSL colour space.

Returns

A ValueType containing the H, S and L components of the Colour. Each component has range [0, 1].

Definition at line 535 of file Colour.cs.

7.18.3.6 L

```
double VectSharp.Colour.L
```

Converts a Colour to the CIE Lab colour space (under Illuminant D65).

Returns

A ValueType containing the L$*$, a$*$ and b$*$ components of the Colour.

Definition at line 466 of file Colour.cs.

7.18.3.7 R

```
double VectSharp.Colour.R
```

Red component of the colour. Range: [0, 1].

Definition at line 30 of file Colour.cs.

7.18.3.8 S

```
double double VectSharp.Colour.S
```

Definition at line 535 of file Colour.cs.

7.18.3.9 X

```
double VectSharp.Colour.X
```

Converts a Colour to the CIE XYZ colour space.

Returns

A ValueTuple containing the X, Y and Z components of the Colour.

Definition at line 377 of file Colour.cs.

7.18.3.10 Y

```
double double VectSharp.Colour.Y
```

Definition at line 377 of file Colour.cs.

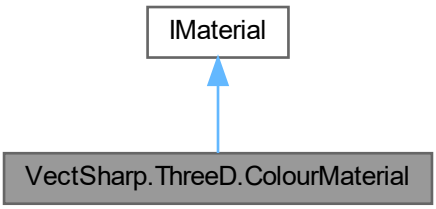
The documentation for this struct was generated from the following files:

- VectSharp/Colour.cs
- VectSharp/StandardColours.cs

7.19 VectSharp.ThreeD.ColourMaterial Class Reference

Represents a material that always has the same colour, regardless of light.

Inheritance diagram for VectSharp.ThreeD.ColourMaterial:



Public Member Functions

- ColourMaterial (Colour colour)

  Creates a new ColourMaterial instance.
- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

Parameters

point	*The point whose colour should be determined.*
surfaceNormal	*The normal to the surface at the specified point .*
camera	*The camera being used to render the scene.*
lights	*A list of light sources that are present in the scene.*
obstructions	*A list of values indicating how obstructed each light source is.*

Returns

The Colour of the specified point.

Properties

- Colour Colour `[get]`

  The colour of the material.

7.19.1 Detailed Description

Represents a material that always has the same colour, regardless of light.

Definition at line 48 of file Materials.cs.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 ColourMaterial()

```
VectSharp.ThreeD.ColourMaterial.ColourMaterial (
            Colour colour )
```

Creates a new ColourMaterial instance.

Parameters

colour	The colour of the material.

Definition at line 59 of file Materials.cs.

7.19.3 Member Function Documentation

7.19.3.1 GetColour()

```
Colour VectSharp.ThreeD.ColourMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implements VectSharp.ThreeD.IMaterial.

Definition at line 65 of file Materials.cs.

7.19.4 Property Documentation

7.19.4.1 Colour

```
Colour VectSharp.ThreeD.ColourMaterial.Colour  [get]
```

The colour of the material.

Definition at line 53 of file Materials.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.20 VectSharp.Filters.ColourMatrix Class Reference

Represents a colour transformation matrix.

Public Member Functions

- ColourMatrix WithAlpha (double alpha)

  Creates a new ColourMatrix whose alpha coefficients are multiplied by the specified value.
- ColourMatrix (double[,] matrix)

  Creates a new ColourMatrix with the specified coefficients.
- Colour Apply (Colour colour)

  Applies the ColourMatrix to the specified Colour.
- void Apply (ref byte R, ref byte G, ref byte B, ref byte A)

  Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the same variables as the original RGBA values.
- void Apply (ref byte R, ref byte G, ref byte B)

  Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the same variables as the original RGB values.
- void Apply (byte R, byte G, byte B, byte A, out byte r, out byte g, out byte b, out byte a)

  Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the specified output bytes.
- void Apply (byte R, byte G, byte B, out byte r, out byte g, out byte b)

  Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the specified output bytes.

Static Public Member Functions

- static ColourMatrix ToColour (Colour colour, bool useAlpha=false)

  Creates a ColourMatrix that turns every colour to which it is applied into the specified colour .
- static ColourMatrix LuminanceToColour (Colour colour, bool useAlpha=false)

  Creates a ColourMatrix that turns every colour to which it is applied into a shade of the specified colour . The brightness of the output colour depends on the luminance of the input colour.
- static ColourMatrix LuminanceToAlpha (bool preserveColour=false)

  Creates a ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.
- static ColourMatrix operator∗ (ColourMatrix matrix1, ColourMatrix matrix2)

  Concatenates two matrices. The resulting ColourMatrix is equivalent to first applying matrix2 , and then matrix1 .

Static Public Attributes

- static ColourMatrix Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that whose output colour is always the same as the input colour.
- static ColourMatrix GreyScale = new ColourMatrix(new double[,] { { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that transforms every colour in a shade of grey with approximately the same luminance.
- static ColourMatrix Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0 }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix producing a "pastel" (desaturation) effect.
- static ColourMatrix Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, { 0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that inverts every colour, leaving the alpha component intact.
- static ColourMatrix AlphaInversion = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that inverts the alpha component, leaving the other components intact.
- static ColourMatrix InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0, -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that shifts every colour component by an amount corresponding to the inverted alpha value. The alpha component is left intact.

Properties

- double R1 `[get, set]`

  The coefficient relating the R component of the output colour to the R component of the input colour.
- double R2 `[get, set]`

  The coefficient relating the R component of the output colour to the G component of the input colour.
- double R3 `[get, set]`

  The coefficient relating the R component of the output colour to the B component of the input colour.
- double R4 `[get, set]`

  The coefficient relating the R component of the output colour to the A component of the input colour.
- double R5 `[get, set]`

  The bias (translation) applied to the R component of the output colour.
- double G1 `[get, set]`

  The coefficient relating the G component of the output colour to the R component of the input colour.
- double G2 `[get, set]`

  The coefficient relating the G component of the output colour to the G component of the input colour.
- double G3 `[get, set]`

  The coefficient relating the G component of the output colour to the B component of the input colour.
- double G4 `[get, set]`

  The coefficient relating the G component of the output colour to the A component of the input colour.
- double G5 `[get, set]`

  The bias (translation) applied to the R component of the output colour.
- double B1 `[get, set]`

  The coefficient relating the B component of the output colour to the R component of the input colour.
- double B2 `[get, set]`

  The coefficient relating the B component of the output colour to the G component of the input colour.
- double B3 `[get, set]`

  The coefficient relating the B component of the output colour to the B component of the input colour.
- double B4 `[get, set]`

The coefficient relating the B component of the output colour to the A component of the input colour.

- double B5 `[get, set]`

  The bias (translation) applied to the B component of the output colour.

- double A1 `[get, set]`

  The coefficient relating the A component of the output colour to the R component of the input colour.

- double A2 `[get, set]`

  The coefficient relating the A component of the output colour to the G component of the input colour.

- double A3 `[get, set]`

  The coefficient relating the A component of the output colour to the B component of the input colour.

- double A4 `[get, set]`

  The coefficient relating the A component of the output colour to the A component of the input colour.

- double A5 `[get, set]`

  The bias (translation) applied to the A component of the output colour.

- double this[int y, int x] `[get]`

  Gets or sets the requested element of the matrix. Elements of the last row of the matrix can be read, but not set.

7.20.1 Detailed Description

Represents a colour transformation matrix.

Definition at line 26 of file ColourMatrixFilter.cs.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 ColourMatrix()

```
VectSharp.Filters.ColourMatrix.ColourMatrix (
            double matrix[,] )
```

Creates a new ColourMatrix with the specified coefficients.

Parameters

matrix	The coefficients of the ColourMatrix.

Definition at line 454 of file ColourMatrixFilter.cs.

7.20.3 Member Function Documentation

7.20.3.1 Apply() [1/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            byte R,
            byte G,
            byte B,
            byte A,
            out byte r,
            out byte g,
            out byte b,
            out byte a )
```

Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the specified output bytes.

Parameters

R	The R component of the input colour.
G	The G component of the input colour.
B	The B component of the input colour.
A	The A component of the input colour.
r	After this method returns, this will contain the R component of the output colour.
g	After this method returns, this will contain the G component of the output colour.
b	After this method returns, this will contain the B component of the output colour.
a	After this method returns, this will contain the A component of the output colour.

Definition at line 538 of file ColourMatrixFilter.cs.

7.20.3.2 Apply() [2/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            byte R,
            byte G,
            byte B,
            out byte r,
            out byte g,
            out byte b )
```

Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the specified output bytes.

Parameters

R	The R component of the input colour.
G	The G component of the input colour.
B	The B component of the input colour.
r	After this method returns, this will contain the R component of the output colour.
g	After this method returns, this will contain the G component of the output colour.
b	After this method returns, this will contain the B component of the output colour.

Definition at line 556 of file ColourMatrixFilter.cs.

7.20.3.3 Apply() [3/5]

```
Colour VectSharp.Filters.ColourMatrix.Apply (
            Colour colour )
```

Applies the ColourMatrix to the specified Colour.

Parameters

colour	The Colour to which the ColourMatrix should be applied.

Returns

The result of applying the ColourMatrix to the specified colour.

Definition at line 470 of file ColourMatrixFilter.cs.

7.20.3.4 Apply() [4/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            ref byte R,
            ref byte G,
            ref byte B )
```

Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the same variables as the original RGB values.

Parameters

R	The R component of the input colour. After this method returns, this will contain the R component of the output colour.
G	The G component of the input colour. After this method returns, this will contain the G component of the output colour.
B	The B component of the input colour. After this method returns, this will contain the B component of the output colour.

Definition at line 515 of file ColourMatrixFilter.cs.

7.20.3.5 Apply() [5/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            ref byte R,
```

```
            ref byte G,
            ref byte B,
            ref byte A )
```

Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the same variables as the original RGBA values.

Parameters

R	The R component of the input colour. After this method returns, this will contain the R component of the output colour.
G	The G component of the input colour. After this method returns, this will contain the G component of the output colour.
B	The B component of the input colour. After this method returns, this will contain the B component of the output colour.
A	The A component of the input colour. After this method returns, this will contain the A component of the output colour.

Definition at line 495 of file ColourMatrixFilter.cs.

7.20.3.6 LuminanceToAlpha()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.LuminanceToAlpha (
            bool preserveColour = false )  [static]
```

Creates a ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.

Parameters

preserveColour	If this is `true`, the values of the red, green and blue components of the input colour are preserved in the output colour. If this is false, the output colour will always be black.

Returns

A ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.

Definition at line 404 of file ColourMatrixFilter.cs.

7.20.3.7 LuminanceToColour()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.LuminanceToColour (
            Colour colour,
            bool useAlpha = false )  [static]
```

Creates a ColourMatrix that turns every colour to which it is applied into a shade of the specified *colour* . The brightness of the output colour depends on the luminance of the input colour.

Parameters

colour	The colour whose shades will be produced by the ColourMatrix.
useAlpha	If this is `true`, the transformation will also be applied to the alpha channel. If this is false, the alpha value of the input pixels is preserved.

Returns

A ColourMatrix that turns every colour to which it is applied into a shade of the specified *colour* .

Definition at line 387 of file ColourMatrixFilter.cs.

7.20.3.8 operator∗()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.operator* (
            ColourMatrix matrix1,
            ColourMatrix matrix2 )  [static]
```

Concatenates two matrices. The resulting ColourMatrix is equivalent to first applying *matrix2* , and then *matrix1* .

Parameters

matrix1	The matrix that acts second.
matrix2	The matrix that acts first.

Returns

A ColourMatrix equivalent to first applying *matrix2* , and then *matrix1* .

Definition at line 432 of file ColourMatrixFilter.cs.

7.20.3.9 ToColour()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.ToColour (
            Colour colour,
            bool useAlpha = false )  [static]
```

Creates a ColourMatrix that turns every colour to which it is applied into the specified *colour* .

Parameters

colour	The colour that will be produced by the ColourMatrix.
useAlpha	If this is `true`, all output pixels will have the same alpha value as the supplied *colour* . If this is false, the alpha value of the input pixels is preserved.

Returns

A ColourMatrix that turns every colour to which it is applied into the specified *colour* .

Definition at line 369 of file ColourMatrixFilter.cs.

7.20.3.10 WithAlpha()

```
ColourMatrix VectSharp.Filters.ColourMatrix.WithAlpha (
            double alpha )
```

Creates a new ColourMatrix whose alpha coefficients are multiplied by the specified value.

Parameters

alpha	The value that will be used to multiply all the alpha coefficients of the ColourMatrix.

Returns

A new ColourMatrix whose alpha coefficients have been multiplied by the specified value.

Definition at line 421 of file ColourMatrixFilter.cs.

7.20.4 Member Data Documentation

7.20.4.1 AlphaInversion

```
ColourMatrix VectSharp.Filters.ColourMatrix.AlphaInversion = new ColourMatrix(new double[,] {
{ 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 }
})  [static]
```

A ColourMatrix that inverts the alpha component, leaving the other components intact.

Definition at line 356 of file ColourMatrixFilter.cs.

7.20.4.2 GreyScale

```
ColourMatrix VectSharp.Filters.ColourMatrix.GreyScale = new ColourMatrix(new double[,] { {
0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0,
0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix that transforms every colour in a shade of grey with approximately the same luminance.

Definition at line 341 of file ColourMatrixFilter.cs.

7.20.4.3 Identity

```
ColourMatrix VectSharp.Filters.ColourMatrix.Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })
[static]
```

A ColourMatrix that whose output colour is always the same as the input colour.

Definition at line 336 of file ColourMatrixFilter.cs.

7.20.4.4 Inversion

```
ColourMatrix VectSharp.Filters.ColourMatrix.Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, { 0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })
[static]
```

A ColourMatrix that inverts every colour, leaving the alpha component intact.

Definition at line 351 of file ColourMatrixFilter.cs.

7.20.4.5 InvertedAlphaShift

```
ColourMatrix VectSharp.Filters.ColourMatrix.InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0, -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix that shifts every colour component by an amount corresponding to the inverted alpha value. The alpha component is left intact.

Definition at line 361 of file ColourMatrixFilter.cs.

7.20.4.6 Pastel

```
ColourMatrix VectSharp.Filters.ColourMatrix.Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0 }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix producing a "pastel" (desaturation) effect.

Definition at line 346 of file ColourMatrixFilter.cs.

7.20.5 Property Documentation

7.20.5.1 A1

```
double VectSharp.Filters.ColourMatrix.A1  [get], [set]
```

The coefficient relating the A component of the output colour to the R component of the input colour.

Definition at line 106 of file ColourMatrixFilter.cs.

7.20.5.2 A2

```
double VectSharp.Filters.ColourMatrix.A2  [get], [set]
```

The coefficient relating the A component of the output colour to the G component of the input colour.

Definition at line 111 of file ColourMatrixFilter.cs.

7.20.5.3 A3

```
double VectSharp.Filters.ColourMatrix.A3  [get], [set]
```

The coefficient relating the A component of the output colour to the B component of the input colour.

Definition at line 116 of file ColourMatrixFilter.cs.

7.20.5.4 A4

```
double VectSharp.Filters.ColourMatrix.A4  [get], [set]
```

The coefficient relating the A component of the output colour to the A component of the input colour.

Definition at line 121 of file ColourMatrixFilter.cs.

7.20.5.5 A5

```
double VectSharp.Filters.ColourMatrix.A5  [get], [set]
```

The bias (translation) applied to the A component of the output colour.

Definition at line 126 of file ColourMatrixFilter.cs.

7.20.5.6 B1

```
double VectSharp.Filters.ColourMatrix.B1  [get], [set]
```

The coefficient relating the B component of the output colour to the R component of the input colour.

Definition at line 81 of file ColourMatrixFilter.cs.

7.20.5.7 B2

```
double VectSharp.Filters.ColourMatrix.B2  [get], [set]
```

The coefficient relating the B component of the output colour to the G component of the input colour.

Definition at line 86 of file ColourMatrixFilter.cs.

7.20.5.8 B3

```
double VectSharp.Filters.ColourMatrix.B3  [get], [set]
```

The coefficient relating the B component of the output colour to the B component of the input colour.

Definition at line 91 of file ColourMatrixFilter.cs.

7.20.5.9 B4

```
double VectSharp.Filters.ColourMatrix.B4  [get], [set]
```

The coefficient relating the B component of the output colour to the A component of the input colour.

Definition at line 96 of file ColourMatrixFilter.cs.

7.20.5.10 B5

```
double VectSharp.Filters.ColourMatrix.B5  [get], [set]
```

The bias (translation) applied to the B component of the output colour.

Definition at line 101 of file ColourMatrixFilter.cs.

7.20.5.11 G1

```
double VectSharp.Filters.ColourMatrix.G1  [get], [set]
```

The coefficient relating the G component of the output colour to the R component of the input colour.

Definition at line 56 of file ColourMatrixFilter.cs.

7.20.5.12 G2

```
double VectSharp.Filters.ColourMatrix.G2  [get], [set]
```

The coefficient relating the G component of the output colour to the G component of the input colour.

Definition at line 61 of file ColourMatrixFilter.cs.

7.20.5.13 G3

```
double VectSharp.Filters.ColourMatrix.G3  [get], [set]
```

The coefficient relating the G component of the output colour to the B component of the input colour.

Definition at line 66 of file ColourMatrixFilter.cs.

7.20.5.14 G4

```
double VectSharp.Filters.ColourMatrix.G4  [get], [set]
```

The coefficient relating the G component of the output colour to the A component of the input colour.

Definition at line 71 of file ColourMatrixFilter.cs.

7.20.5.15 G5

```
double VectSharp.Filters.ColourMatrix.G5  [get], [set]
```

The bias (translation) applied to the R component of the output colour.

Definition at line 76 of file ColourMatrixFilter.cs.

7.20.5.16 R1

```
double VectSharp.Filters.ColourMatrix.R1  [get], [set]
```

The coefficient relating the R component of the output colour to the R component of the input colour.

Definition at line 31 of file ColourMatrixFilter.cs.

7.20.5.17 R2

```
double VectSharp.Filters.ColourMatrix.R2  [get], [set]
```

The coefficient relating the R component of the output colour to the G component of the input colour.

Definition at line 36 of file ColourMatrixFilter.cs.

7.20.5.18 R3

```
double VectSharp.Filters.ColourMatrix.R3  [get], [set]
```

The coefficient relating the R component of the output colour to the B component of the input colour.

Definition at line 41 of file ColourMatrixFilter.cs.

7.20.5.19 R4

```
double VectSharp.Filters.ColourMatrix.R4  [get], [set]
```

The coefficient relating the R component of the output colour to the A component of the input colour.

Definition at line 46 of file ColourMatrixFilter.cs.

7.20.5.20 R5

```
double VectSharp.Filters.ColourMatrix.R5  [get], [set]
```

The bias (translation) applied to the R component of the output colour.

Definition at line 51 of file ColourMatrixFilter.cs.

7.20.5.21 this[int y, int x]

```
double VectSharp.Filters.ColourMatrix.this[int y, int x]  [get]
```

Gets or sets the requested element of the matrix. Elements of the last row of the matrix can be read, but not set.

Parameters

y	The row of the matrix.
x	The column of the matrix.

Returns

The requested element of the matrix.

Exceptions

ArgumentOutOfRangeException	An attempt has been made to access an element out of the bounds of the matrix.

Definition at line 135 of file ColourMatrixFilter.cs.

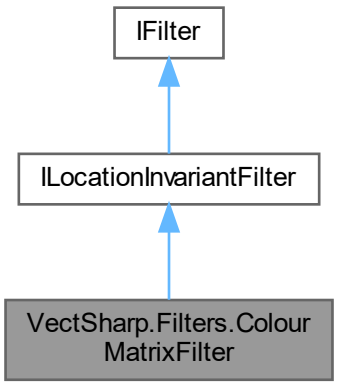
The documentation for this class was generated from the following file:

- VectSharp/Filters/ColourMatrixFilter.cs

7.21 VectSharp.Filters.ColourMatrixFilter Class Reference

Represents a filter that applies a Filters.ColourMatrix to the colours of the image.

Inheritance diagram for VectSharp.Filters.ColourMatrixFilter:



Public Member Functions

- ColourMatrixFilter (ColourMatrix colorMatrix)

  Creates a new ColourMatrixFilter with the specified Filters.ColourMatrix.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- ColourMatrix ColourMatrix `[get]`

  The Filters.ColourMatrix that is applied by this filter.
- Point TopLeftMargin = new Point() `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin = new Point() `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.21.1 Detailed Description

Represents a filter that applies a Filters.ColourMatrix to the colours of the image.

Definition at line 567 of file ColourMatrixFilter.cs.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 ColourMatrixFilter()

```
VectSharp.Filters.ColourMatrixFilter.ColourMatrixFilter (
            ColourMatrix colorMatrix )
```

Creates a new ColourMatrixFilter with the specified Filters.ColourMatrix.

Parameters

colorMatrix	The Filters.ColourMatrix that will be applied by the filter.

Definition at line 584 of file ColourMatrixFilter.cs.

7.21.3 Member Function Documentation

7.21.3.1 Filter()

```
RasterImage VectSharp.Filters.ColourMatrixFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 590 of file ColourMatrixFilter.cs.

7.21.4 Property Documentation

7.21.4.1 BottomRightMargin

```
Point VectSharp.Filters.ColourMatrixFilter.BottomRightMargin = new Point()  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 578 of file ColourMatrixFilter.cs.

7.21.4.2 ColourMatrix

```
ColourMatrix VectSharp.Filters.ColourMatrixFilter.ColourMatrix  [get]
```

The Filters.ColourMatrix that is applied by this filter.

Definition at line 572 of file ColourMatrixFilter.cs.

7.21.4.3 TopLeftMargin

```
Point VectSharp.Filters.ColourMatrixFilter.TopLeftMargin = new Point()  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 575 of file ColourMatrixFilter.cs.

The documentation for this class was generated from the following file:

- VectSharp/Filters/ColourMatrixFilter.cs

7.22 VectSharp.Colours Class Reference

Standard colours.

Static Public Attributes

- static Colour Black = Colour.FromRgb(0, 0, 0)
  Black #000000
- static Colour Navy = Colour.FromRgb(0, 0, 128)
  Navy #000080
- static Colour DarkBlue = Colour.FromRgb(0, 0, 139)
  DarkBlue #00008B
- static Colour MediumBlue = Colour.FromRgb(0, 0, 205)
  MediumBlue #0000CD
- static Colour Blue = Colour.FromRgb(0, 0, 255)
  Blue #0000FF
- static Colour DarkGreen = Colour.FromRgb(0, 100, 0)
  DarkGreen #006400
- static Colour Green = Colour.FromRgb(0, 128, 0)
  Green #008000
- static Colour Teal = Colour.FromRgb(0, 128, 128)
  Teal #008080
- static Colour DarkCyan = Colour.FromRgb(0, 139, 139)
  DarkCyan #008B8B
- static Colour DeepSkyBlue = Colour.FromRgb(0, 191, 255)
  DeepSkyBlue #00BFFF
- static Colour DarkTurquoise = Colour.FromRgb(0, 206, 209)
  DarkTurquoise #00CED1
- static Colour MediumSpringGreen = Colour.FromRgb(0, 250, 154)
  MediumSpringGreen #00FA9A
- static Colour Lime = Colour.FromRgb(0, 255, 0)
  Lime #00FF00
- static Colour SpringGreen = Colour.FromRgb(0, 255, 127)
  SpringGreen #00FF7F

- static Colour Aqua = Colour.FromRgb(0, 255, 255)

  Aqua #00FFFF
- static Colour Cyan = Colour.FromRgb(0, 255, 255)

  Cyan #00FFFF
- static Colour MidnightBlue = Colour.FromRgb(25, 25, 112)

  MidnightBlue #191970
- static Colour DodgerBlue = Colour.FromRgb(30, 144, 255)

  DodgerBlue #1E90FF
- static Colour LightSeaGreen = Colour.FromRgb(32, 178, 170)

  LightSeaGreen #20B2AA
- static Colour ForestGreen = Colour.FromRgb(34, 139, 34)

  ForestGreen #228B22
- static Colour SeaGreen = Colour.FromRgb(46, 139, 87)

  SeaGreen #2E8B57
- static Colour DarkSlateGray = Colour.FromRgb(47, 79, 79)

  DarkSlateGray #2F4F4F
- static Colour DarkSlateGrey = Colour.FromRgb(47, 79, 79)

  DarkSlateGrey #2F4F4F
- static Colour LimeGreen = Colour.FromRgb(50, 205, 50)

  LimeGreen #32CD32
- static Colour MediumSeaGreen = Colour.FromRgb(60, 179, 113)

  MediumSeaGreen #3CB371
- static Colour Turquoise = Colour.FromRgb(64, 224, 208)

  Turquoise #40E0D0
- static Colour RoyalBlue = Colour.FromRgb(65, 105, 225)

  RoyalBlue #4169E1
- static Colour SteelBlue = Colour.FromRgb(70, 130, 180)

  SteelBlue #4682B4
- static Colour DarkSlateBlue = Colour.FromRgb(72, 61, 139)

  DarkSlateBlue #483D8B
- static Colour MediumTurquoise = Colour.FromRgb(72, 209, 204)

  MediumTurquoise #48D1CC
- static Colour Indigo = Colour.FromRgb(75, 0, 130)

  Indigo #4B0082
- static Colour DarkOliveGreen = Colour.FromRgb(85, 107, 47)

  DarkOliveGreen #556B2F
- static Colour CadetBlue = Colour.FromRgb(95, 158, 160)

  CadetBlue #5F9EA0
- static Colour CornflowerBlue = Colour.FromRgb(100, 149, 237)

  CornflowerBlue #6495ED
- static Colour RebeccaPurple = Colour.FromRgb(102, 51, 153)

  RebeccaPurple #663399
- static Colour MediumAquaMarine = Colour.FromRgb(102, 205, 170)

  MediumAquaMarine #66CDAA
- static Colour DimGray = Colour.FromRgb(105, 105, 105)

  DimGray #696969
- static Colour DimGrey = Colour.FromRgb(105, 105, 105)

  DimGrey #696969
- static Colour SlateBlue = Colour.FromRgb(106, 90, 205)

  SlateBlue #6A5ACD
- static Colour OliveDrab = Colour.FromRgb(107, 142, 35)

OliveDrab #6B8E23

- static Colour SlateGray = Colour.FromRgb(112, 128, 144)

  SlateGray #708090
- static Colour SlateGrey = Colour.FromRgb(112, 128, 144)

  SlateGrey #708090
- static Colour LightSlateGray = Colour.FromRgb(119, 136, 153)

  LightSlateGray #778899
- static Colour LightSlateGrey = Colour.FromRgb(119, 136, 153)

  LightSlateGrey #778899
- static Colour MediumSlateBlue = Colour.FromRgb(123, 104, 238)

  MediumSlateBlue #7B68EE
- static Colour LawnGreen = Colour.FromRgb(124, 252, 0)

  LawnGreen #7CFC00
- static Colour Chartreuse = Colour.FromRgb(127, 255, 0)

  Chartreuse #7FFF00
- static Colour Aquamarine = Colour.FromRgb(127, 255, 212)

  Aquamarine #7FFFD4
- static Colour Maroon = Colour.FromRgb(128, 0, 0)

  Maroon #800000
- static Colour Purple = Colour.FromRgb(128, 0, 128)

  Purple #800080
- static Colour Olive = Colour.FromRgb(128, 128, 0)

  Olive #808000
- static Colour Gray = Colour.FromRgb(128, 128, 128)

  Gray #808080
- static Colour Grey = Colour.FromRgb(128, 128, 128)

  Grey #808080
- static Colour SkyBlue = Colour.FromRgb(135, 206, 235)

  SkyBlue #87CEEB
- static Colour LightSkyBlue = Colour.FromRgb(135, 206, 250)

  LightSkyBlue #87CEFA
- static Colour BlueViolet = Colour.FromRgb(138, 43, 226)

  BlueViolet #8A2BE2
- static Colour DarkRed = Colour.FromRgb(139, 0, 0)

  DarkRed #8B0000
- static Colour DarkMagenta = Colour.FromRgb(139, 0, 139)

  DarkMagenta #8B008B
- static Colour SaddleBrown = Colour.FromRgb(139, 69, 19)

  SaddleBrown #8B4513
- static Colour DarkSeaGreen = Colour.FromRgb(143, 188, 143)

  DarkSeaGreen #8FBC8F
- static Colour LightGreen = Colour.FromRgb(144, 238, 144)

  LightGreen #90EE90
- static Colour MediumPurple = Colour.FromRgb(147, 112, 219)

  MediumPurple #9370DB
- static Colour DarkViolet = Colour.FromRgb(148, 0, 211)

  DarkViolet #9400D3
- static Colour PaleGreen = Colour.FromRgb(152, 251, 152)

  PaleGreen #98FB98
- static Colour DarkOrchid = Colour.FromRgb(153, 50, 204)

  DarkOrchid #9932CC

- static Colour YellowGreen = Colour.FromRgb(154, 205, 50)

  YellowGreen #9ACD32
- static Colour Sienna = Colour.FromRgb(160, 82, 45)

  Sienna #A0522D
- static Colour Brown = Colour.FromRgb(165, 42, 42)

  Brown #A52A2A
- static Colour DarkGray = Colour.FromRgb(169, 169, 169)

  DarkGray #A9A9A9
- static Colour DarkGrey = Colour.FromRgb(169, 169, 169)

  DarkGrey #A9A9A9
- static Colour LightBlue = Colour.FromRgb(173, 216, 230)

  LightBlue #ADD8E6
- static Colour GreenYellow = Colour.FromRgb(173, 255, 47)

  GreenYellow #ADFF2F
- static Colour PaleTurquoise = Colour.FromRgb(175, 238, 238)

  PaleTurquoise #AFEEEE
- static Colour LightSteelBlue = Colour.FromRgb(176, 196, 222)

  LightSteelBlue #B0C4DE
- static Colour PowderBlue = Colour.FromRgb(176, 224, 230)

  PowderBlue #B0E0E6
- static Colour FireBrick = Colour.FromRgb(178, 34, 34)

  FireBrick #B22222
- static Colour DarkGoldenRod = Colour.FromRgb(184, 134, 11)

  DarkGoldenRod #B8860B
- static Colour MediumOrchid = Colour.FromRgb(186, 85, 211)

  MediumOrchid #BA55D3
- static Colour RosyBrown = Colour.FromRgb(188, 143, 143)

  RosyBrown #BC8F8F
- static Colour DarkKhaki = Colour.FromRgb(189, 183, 107)

  DarkKhaki #BDB76B
- static Colour Silver = Colour.FromRgb(192, 192, 192)

  Silver #C0C0C0
- static Colour MediumVioletRed = Colour.FromRgb(199, 21, 133)

  MediumVioletRed #C71585
- static Colour IndianRed = Colour.FromRgb(205, 92, 92)

  IndianRed #CD5C5C
- static Colour Peru = Colour.FromRgb(205, 133, 63)

  Peru #CD853F
- static Colour Chocolate = Colour.FromRgb(210, 105, 30)

  Chocolate #D2691E
- static Colour Tan = Colour.FromRgb(210, 180, 140)

  Tan #D2B48C
- static Colour LightGray = Colour.FromRgb(211, 211, 211)

  LightGray #D3D3D3
- static Colour LightGrey = Colour.FromRgb(211, 211, 211)

  LightGrey #D3D3D3
- static Colour Thistle = Colour.FromRgb(216, 191, 216)

  Thistle #D8BFD8
- static Colour Orchid = Colour.FromRgb(218, 112, 214)

  Orchid #DA70D6
- static Colour GoldenRod = Colour.FromRgb(218, 165, 32)

GoldenRod #DAA520

- static Colour PaleVioletRed = Colour.FromRgb(219, 112, 147)

  PaleVioletRed #DB7093
- static Colour Crimson = Colour.FromRgb(220, 20, 60)

  Crimson #DC143C
- static Colour Gainsboro = Colour.FromRgb(220, 220, 220)

  Gainsboro #DCDCDC
- static Colour Plum = Colour.FromRgb(221, 160, 221)

  Plum #DDA0DD
- static Colour BurlyWood = Colour.FromRgb(222, 184, 135)

  BurlyWood #DEB887
- static Colour LightCyan = Colour.FromRgb(224, 255, 255)

  LightCyan #E0FFFF
- static Colour Lavender = Colour.FromRgb(230, 230, 250)

  Lavender #E6E6FA
- static Colour DarkSalmon = Colour.FromRgb(233, 150, 122)

  DarkSalmon #E9967A
- static Colour Violet = Colour.FromRgb(238, 130, 238)

  Violet #EE82EE
- static Colour PaleGoldenRod = Colour.FromRgb(238, 232, 170)

  PaleGoldenRod #EEE8AA
- static Colour LightCoral = Colour.FromRgb(240, 128, 128)

  LightCoral #F08080
- static Colour Khaki = Colour.FromRgb(240, 230, 140)

  Khaki #F0E68C
- static Colour AliceBlue = Colour.FromRgb(240, 248, 255)

  AliceBlue #F0F8FF
- static Colour HoneyDew = Colour.FromRgb(240, 255, 240)

  HoneyDew #F0FFF0
- static Colour Azure = Colour.FromRgb(240, 255, 255)

  Azure #F0FFFF
- static Colour SandyBrown = Colour.FromRgb(244, 164, 96)

  SandyBrown #F4A460
- static Colour Wheat = Colour.FromRgb(245, 222, 179)

  Wheat #F5DEB3
- static Colour Beige = Colour.FromRgb(245, 245, 220)

  Beige #F5F5DC
- static Colour WhiteSmoke = Colour.FromRgb(245, 245, 245)

  WhiteSmoke #F5F5F5
- static Colour MintCream = Colour.FromRgb(245, 255, 250)

  MintCream #F5FFFA
- static Colour GhostWhite = Colour.FromRgb(248, 248, 255)

  GhostWhite #F8F8FF
- static Colour Salmon = Colour.FromRgb(250, 128, 114)

  Salmon #FA8072
- static Colour AntiqueWhite = Colour.FromRgb(250, 235, 215)

  AntiqueWhite #FAEBD7
- static Colour Linen = Colour.FromRgb(250, 240, 230)

  Linen #FAF0E6
- static Colour LightGoldenRodYellow = Colour.FromRgb(250, 250, 210)

  LightGoldenRodYellow #FAFAD2

- static Colour OldLace = Colour.FromRgb(253, 245, 230)

  OldLace #FDF5E6
- static Colour Red = Colour.FromRgb(255, 0, 0)

  Red #FF0000
- static Colour Fuchsia = Colour.FromRgb(255, 0, 255)

  Fuchsia #FF00FF
- static Colour Magenta = Colour.FromRgb(255, 0, 255)

  Magenta #FF00FF
- static Colour DeepPink = Colour.FromRgb(255, 20, 147)

  DeepPink #FF1493
- static Colour OrangeRed = Colour.FromRgb(255, 69, 0)

  OrangeRed #FF4500
- static Colour Tomato = Colour.FromRgb(255, 99, 71)

  Tomato #FF6347
- static Colour HotPink = Colour.FromRgb(255, 105, 180)

  HotPink #FF69B4
- static Colour Coral = Colour.FromRgb(255, 127, 80)

  Coral #FF7F50
- static Colour DarkOrange = Colour.FromRgb(255, 140, 0)

  DarkOrange #FF8C00
- static Colour LightSalmon = Colour.FromRgb(255, 160, 122)

  LightSalmon #FFA07A
- static Colour Orange = Colour.FromRgb(255, 165, 0)

  Orange #FFA500
- static Colour LightPink = Colour.FromRgb(255, 182, 193)

  LightPink #FFB6C1
- static Colour Pink = Colour.FromRgb(255, 192, 203)

  Pink #FFC0CB
- static Colour Gold = Colour.FromRgb(255, 215, 0)

  Gold #FFD700
- static Colour PeachPuff = Colour.FromRgb(255, 218, 185)

  PeachPuff #FFDAB9
- static Colour NavajoWhite = Colour.FromRgb(255, 222, 173)

  NavajoWhite #FFDEAD
- static Colour Moccasin = Colour.FromRgb(255, 228, 181)

  Moccasin #FFE4B5
- static Colour Bisque = Colour.FromRgb(255, 228, 196)

  Bisque #FFE4C4
- static Colour MistyRose = Colour.FromRgb(255, 228, 225)

  MistyRose #FFE4E1
- static Colour BlanchedAlmond = Colour.FromRgb(255, 235, 205)

  BlanchedAlmond #FFEBCD
- static Colour PapayaWhip = Colour.FromRgb(255, 239, 213)

  PapayaWhip #FFEFD5
- static Colour LavenderBlush = Colour.FromRgb(255, 240, 245)

  LavenderBlush #FFF0F5
- static Colour SeaShell = Colour.FromRgb(255, 245, 238)

  SeaShell #FFF5EE
- static Colour Cornsilk = Colour.FromRgb(255, 248, 220)

  Cornsilk #FFF8DC
- static Colour LemonChiffon = Colour.FromRgb(255, 250, 205)

LemonChiffon #FFFACD

- static Colour FloralWhite = Colour.FromRgb(255, 250, 240)

  FloralWhite #FFFAF0
- static Colour Snow = Colour.FromRgb(255, 250, 250)

  Snow #FFFAFA
- static Colour Yellow = Colour.FromRgb(255, 255, 0)

  Yellow #FFFF00
- static Colour LightYellow = Colour.FromRgb(255, 255, 224)

  LightYellow #FFFFE0
- static Colour Ivory = Colour.FromRgb(255, 255, 240)

  Ivory #FFFFF0
- static Colour White = Colour.FromRgb(255, 255, 255)

  White #FFFFFF

7.22.1 Detailed Description

Standard colours.

Definition at line 182 of file StandardColours.cs.

7.22.2 Member Data Documentation

7.22.2.1 AliceBlue

```
Colour VectSharp.Colours.AliceBlue = Colour.FromRgb(240, 248, 255)  [static]
```

AliceBlue #F0F8FF

Definition at line 599 of file StandardColours.cs.

7.22.2.2 AntiqueWhite

```
Colour VectSharp.Colours.AntiqueWhite = Colour.FromRgb(250, 235, 215)  [static]
```

AntiqueWhite #FAEBD7

Definition at line 639 of file StandardColours.cs.

7.22.2.3 Aqua

```
Colour VectSharp.Colours.Aqua = Colour.FromRgb(0, 255, 255)  [static]
```

Aqua #00FFFF

Definition at line 243 of file StandardColours.cs.

7.22.2.4 Aquamarine

```
Colour VectSharp.Colours.Aquamarine = Colour.FromRgb(127, 255, 212)  [static]
```

Aquamarine #7FFFD4

Definition at line 375 of file StandardColours.cs.

7.22.2.5 Azure

```
Colour VectSharp.Colours.Azure = Colour.FromRgb(240, 255, 255)  [static]
```

Azure #F0FFFF

Definition at line 607 of file StandardColours.cs.

7.22.2.6 Beige

```
Colour VectSharp.Colours.Beige = Colour.FromRgb(245, 245, 220)  [static]
```

Beige #F5F5DC

Definition at line 619 of file StandardColours.cs.

7.22.2.7 Bisque

```
Colour VectSharp.Colours.Bisque = Colour.FromRgb(255, 228, 196)  [static]
```

Bisque #FFE4C4

Definition at line 723 of file StandardColours.cs.

7.22.2.8 Black

```
Colour VectSharp.Colours.Black = Colour.FromRgb(0, 0, 0)  [static]
```

Black #000000

Definition at line 187 of file StandardColours.cs.

7.22.2.9 BlanchedAlmond

```
Colour VectSharp.Colours.BlanchedAlmond = Colour.FromRgb(255, 235, 205)  [static]
```

BlanchedAlmond #FFEBCD

Definition at line 731 of file StandardColours.cs.

7.22.2.10 Blue

```
Colour VectSharp.Colours.Blue = Colour.FromRgb(0, 0, 255)  [static]
```

Blue #0000FF

Definition at line 203 of file StandardColours.cs.

7.22.2.11 BlueViolet

```
Colour VectSharp.Colours.BlueViolet = Colour.FromRgb(138, 43, 226)  [static]
```

BlueViolet #8A2BE2

Definition at line 407 of file StandardColours.cs.

7.22.2.12 Brown

```
Colour VectSharp.Colours.Brown = Colour.FromRgb(165, 42, 42)  [static]
```

Brown #A52A2A

Definition at line 455 of file StandardColours.cs.

7.22.2.13 BurlyWood

```
Colour VectSharp.Colours.BurlyWood = Colour.FromRgb(222, 184, 135)  [static]
```

BurlyWood #DEB887

Definition at line 567 of file StandardColours.cs.

7.22.2.14 CadetBlue

```
Colour VectSharp.Colours.CadetBlue = Colour.FromRgb(95, 158, 160)  [static]
```

CadetBlue #5F9EA0

Definition at line 315 of file StandardColours.cs.

7.22.2.15 Chartreuse

```
Colour VectSharp.Colours.Chartreuse = Colour.FromRgb(127, 255, 0)  [static]
```

Chartreuse #7FFF00

Definition at line 371 of file StandardColours.cs.

7.22.2.16 Chocolate

```
Colour VectSharp.Colours.Chocolate = Colour.FromRgb(210, 105, 30)  [static]
```

Chocolate #D2691E

Definition at line 523 of file StandardColours.cs.

7.22.2.17 Coral

```
Colour VectSharp.Colours.Coral = Colour.FromRgb(255, 127, 80)  [static]
```

Coral #FF7F50

Definition at line 683 of file StandardColours.cs.

7.22.2.18 CornflowerBlue

```
Colour VectSharp.Colours.CornflowerBlue = Colour.FromRgb(100, 149, 237)  [static]
```

CornflowerBlue #6495ED

Definition at line 319 of file StandardColours.cs.

7.22.2.19 Cornsilk

```
Colour VectSharp.Colours.Cornsilk = Colour.FromRgb(255, 248, 220)  [static]
```

Cornsilk #FFF8DC

Definition at line 747 of file StandardColours.cs.

7.22.2.20 Crimson

```
Colour VectSharp.Colours.Crimson = Colour.FromRgb(220, 20, 60)  [static]
```

Crimson #DC143C

Definition at line 555 of file StandardColours.cs.

7.22.2.21 Cyan

```
Colour VectSharp.Colours.Cyan = Colour.FromRgb(0, 255, 255)  [static]
```

Cyan #00FFFF

Definition at line 247 of file StandardColours.cs.

7.22.2.22 DarkBlue

```
Colour VectSharp.Colours.DarkBlue = Colour.FromRgb(0, 0, 139)  [static]
```

DarkBlue #00008B

Definition at line 195 of file StandardColours.cs.

7.22.2.23 DarkCyan

```
Colour VectSharp.Colours.DarkCyan = Colour.FromRgb(0, 139, 139)  [static]
```

DarkCyan #008B8B

Definition at line 219 of file StandardColours.cs.

7.22.2.24 DarkGoldenRod

```
Colour VectSharp.Colours.DarkGoldenRod = Colour.FromRgb(184, 134, 11)  [static]
```

DarkGoldenRod #B8860B

Definition at line 491 of file StandardColours.cs.

7.22.2.25 DarkGray

```
Colour VectSharp.Colours.DarkGray = Colour.FromRgb(169, 169, 169)  [static]
```

DarkGray #A9A9A9

Definition at line 459 of file StandardColours.cs.

7.22.2.26 DarkGreen

```
Colour VectSharp.Colours.DarkGreen = Colour.FromRgb(0, 100, 0)  [static]
```

DarkGreen #006400

Definition at line 207 of file StandardColours.cs.

7.22.2.27 DarkGrey

```
Colour VectSharp.Colours.DarkGrey = Colour.FromRgb(169, 169, 169)  [static]
```

DarkGrey #A9A9A9

Definition at line 463 of file StandardColours.cs.

7.22.2.28 DarkKhaki

```
Colour VectSharp.Colours.DarkKhaki = Colour.FromRgb(189, 183, 107)  [static]
```

DarkKhaki #BDB76B

Definition at line 503 of file StandardColours.cs.

7.22.2.29 DarkMagenta

```
Colour VectSharp.Colours.DarkMagenta = Colour.FromRgb(139, 0, 139)  [static]
```

DarkMagenta #8B008B

Definition at line 415 of file StandardColours.cs.

7.22.2.30 DarkOliveGreen

```
Colour VectSharp.Colours.DarkOliveGreen = Colour.FromRgb(85, 107, 47)  [static]
```

DarkOliveGreen #556B2F

Definition at line 311 of file StandardColours.cs.

7.22.2.31 DarkOrange

```
Colour VectSharp.Colours.DarkOrange = Colour.FromRgb(255, 140, 0)  [static]
```

DarkOrange #FF8C00

Definition at line 687 of file StandardColours.cs.

7.22.2.32 DarkOrchid

```
Colour VectSharp.Colours.DarkOrchid = Colour.FromRgb(153, 50, 204)  [static]
```

DarkOrchid #9932CC

Definition at line 443 of file StandardColours.cs.

7.22.2.33 DarkRed

```
Colour VectSharp.Colours.DarkRed = Colour.FromRgb(139, 0, 0)  [static]
```

DarkRed #8B0000

Definition at line 411 of file StandardColours.cs.

7.22.2.34 DarkSalmon

```
Colour VectSharp.Colours.DarkSalmon = Colour.FromRgb(233, 150, 122)  [static]
```

DarkSalmon #E9967A

Definition at line 579 of file StandardColours.cs.

7.22.2.35 DarkSeaGreen

```
Colour VectSharp.Colours.DarkSeaGreen = Colour.FromRgb(143, 188, 143)  [static]
```

DarkSeaGreen #8FBC8F

Definition at line 423 of file StandardColours.cs.

7.22.2.36 DarkSlateBlue

```
Colour VectSharp.Colours.DarkSlateBlue = Colour.FromRgb(72, 61, 139)  [static]
```

DarkSlateBlue #483D8B

Definition at line 299 of file StandardColours.cs.

7.22.2.37 DarkSlateGray

```
Colour VectSharp.Colours.DarkSlateGray = Colour.FromRgb(47, 79, 79)  [static]
```

DarkSlateGray #2F4F4F

Definition at line 271 of file StandardColours.cs.

7.22.2.38 DarkSlateGrey

```
Colour VectSharp.Colours.DarkSlateGrey = Colour.FromRgb(47, 79, 79)  [static]
```

DarkSlateGrey #2F4F4F

Definition at line 275 of file StandardColours.cs.

7.22.2.39 DarkTurquoise

```
Colour VectSharp.Colours.DarkTurquoise = Colour.FromRgb(0, 206, 209)  [static]
```

DarkTurquoise #00CED1

Definition at line 227 of file StandardColours.cs.

7.22.2.40 DarkViolet

```
Colour VectSharp.Colours.DarkViolet = Colour.FromRgb(148, 0, 211)  [static]
```

DarkViolet #9400D3

Definition at line 435 of file StandardColours.cs.

7.22.2.41 DeepPink

```
Colour VectSharp.Colours.DeepPink = Colour.FromRgb(255, 20, 147)  [static]
```

DeepPink #FF1493

Definition at line 667 of file StandardColours.cs.

7.22.2.42 DeepSkyBlue

```
Colour VectSharp.Colours.DeepSkyBlue = Colour.FromRgb(0, 191, 255)  [static]
```

DeepSkyBlue #00BFFF

Definition at line 223 of file StandardColours.cs.

7.22.2.43 DimGray

```
Colour VectSharp.Colours.DimGray = Colour.FromRgb(105, 105, 105)  [static]
```

DimGray #696969

Definition at line 331 of file StandardColours.cs.

7.22.2.44 DimGrey

```
Colour VectSharp.Colours.DimGrey = Colour.FromRgb(105, 105, 105)  [static]
```

DimGrey #696969

Definition at line 335 of file StandardColours.cs.

7.22.2.45 DodgerBlue

```
Colour VectSharp.Colours.DodgerBlue = Colour.FromRgb(30, 144, 255)  [static]
```

DodgerBlue #1E90FF

Definition at line 255 of file StandardColours.cs.

7.22.2.46 FireBrick

```
Colour VectSharp.Colours.FireBrick = Colour.FromRgb(178, 34, 34)  [static]
```

FireBrick #B22222

Definition at line 487 of file StandardColours.cs.

7.22.2.47 FloralWhite

```
Colour VectSharp.Colours.FloralWhite = Colour.FromRgb(255, 250, 240)  [static]
```

FloralWhite #FFFAF0

Definition at line 755 of file StandardColours.cs.

7.22.2.48 ForestGreen

```
Colour VectSharp.Colours.ForestGreen = Colour.FromRgb(34, 139, 34)  [static]
```

ForestGreen #228B22

Definition at line 263 of file StandardColours.cs.

7.22.2.49 Fuchsia

```
Colour VectSharp.Colours.Fuchsia = Colour.FromRgb(255, 0, 255)  [static]
```

Fuchsia #FF00FF

Definition at line 659 of file StandardColours.cs.

7.22.2.50 Gainsboro

```
Colour VectSharp.Colours.Gainsboro = Colour.FromRgb(220, 220, 220)  [static]
```

Gainsboro #DCDCDC

Definition at line 559 of file StandardColours.cs.

7.22.2.51 GhostWhite

```
Colour VectSharp.Colours.GhostWhite = Colour.FromRgb(248, 248, 255)  [static]
```

GhostWhite #F8F8FF

Definition at line 631 of file StandardColours.cs.

7.22.2.52 Gold

```
Colour VectSharp.Colours.Gold = Colour.FromRgb(255, 215, 0)  [static]
```

Gold #FFD700

Definition at line 707 of file StandardColours.cs.

7.22.2.53 GoldenRod

```
Colour VectSharp.Colours.GoldenRod = Colour.FromRgb(218, 165, 32)  [static]
```

GoldenRod #DAA520

Definition at line 547 of file StandardColours.cs.

7.22.2.54 Gray

```
Colour VectSharp.Colours.Gray = Colour.FromRgb(128, 128, 128)  [static]
```

Gray #808080

Definition at line 391 of file StandardColours.cs.

7.22.2.55 Green

```
Colour VectSharp.Colours.Green = Colour.FromRgb(0, 128, 0)  [static]
```

Green #008000

Definition at line 211 of file StandardColours.cs.

7.22.2.56 GreenYellow

```
Colour VectSharp.Colours.GreenYellow = Colour.FromRgb(173, 255, 47)  [static]
```

GreenYellow #ADFF2F

Definition at line 471 of file StandardColours.cs.

7.22.2.57 Grey

```
Colour VectSharp.Colours.Grey = Colour.FromRgb(128, 128, 128)  [static]
```

Grey #808080

Definition at line 395 of file StandardColours.cs.

7.22.2.58 HoneyDew

```
Colour VectSharp.Colours.HoneyDew = Colour.FromRgb(240, 255, 240)  [static]
```

HoneyDew #F0FFF0

Definition at line 603 of file StandardColours.cs.

7.22.2.59 HotPink

```
Colour VectSharp.Colours.HotPink = Colour.FromRgb(255, 105, 180)  [static]
```

HotPink #FF69B4

Definition at line 679 of file StandardColours.cs.

7.22.2.60 IndianRed

```
Colour VectSharp.Colours.IndianRed = Colour.FromRgb(205, 92, 92)  [static]
```

IndianRed #CD5C5C

Definition at line 515 of file StandardColours.cs.

7.22.2.61 Indigo

```
Colour VectSharp.Colours.Indigo = Colour.FromRgb(75, 0, 130)  [static]
```

Indigo #4B0082

Definition at line 307 of file StandardColours.cs.

7.22.2.62 Ivory

```
Colour VectSharp.Colours.Ivory = Colour.FromRgb(255, 255, 240)  [static]
```

Ivory #FFFFF0

Definition at line 771 of file StandardColours.cs.

7.22.2.63 Khaki

```
Colour VectSharp.Colours.Khaki = Colour.FromRgb(240, 230, 140)  [static]
```

Khaki #F0E68C

Definition at line 595 of file StandardColours.cs.

7.22.2.64 Lavender

```
Colour VectSharp.Colours.Lavender = Colour.FromRgb(230, 230, 250)  [static]
```

Lavender #E6E6FA

Definition at line 575 of file StandardColours.cs.

7.22.2.65 LavenderBlush

```
Colour VectSharp.Colours.LavenderBlush = Colour.FromRgb(255, 240, 245)  [static]
```

LavenderBlush #FFF0F5

Definition at line 739 of file StandardColours.cs.

7.22.2.66 LawnGreen

```
Colour VectSharp.Colours.LawnGreen = Colour.FromRgb(124, 252, 0)  [static]
```

LawnGreen #7CFC00

Definition at line 367 of file StandardColours.cs.

7.22.2.67 LemonChiffon

```
Colour VectSharp.Colours.LemonChiffon = Colour.FromRgb(255, 250, 205)  [static]
```

LemonChiffon #FFFACD

Definition at line 751 of file StandardColours.cs.

7.22.2.68 LightBlue

```
Colour VectSharp.Colours.LightBlue = Colour.FromRgb(173, 216, 230)  [static]
```

LightBlue #ADD8E6

Definition at line 467 of file StandardColours.cs.

7.22.2.69 LightCoral

```
Colour VectSharp.Colours.LightCoral = Colour.FromRgb(240, 128, 128)  [static]
```

LightCoral #F08080

Definition at line 591 of file StandardColours.cs.

7.22.2.70 LightCyan

```
Colour VectSharp.Colours.LightCyan = Colour.FromRgb(224, 255, 255)  [static]
```

LightCyan #E0FFFF

Definition at line 571 of file StandardColours.cs.

7.22.2.71 LightGoldenRodYellow

```
Colour VectSharp.Colours.LightGoldenRodYellow = Colour.FromRgb(250, 250, 210)  [static]
```

LightGoldenRodYellow #FAFAD2

Definition at line 647 of file StandardColours.cs.

7.22.2.72 LightGray

```
Colour VectSharp.Colours.LightGray = Colour.FromRgb(211, 211, 211)  [static]
```

LightGray #D3D3D3

Definition at line 531 of file StandardColours.cs.

7.22.2.73 LightGreen

```
Colour VectSharp.Colours.LightGreen = Colour.FromRgb(144, 238, 144)  [static]
```

LightGreen #90EE90

Definition at line 427 of file StandardColours.cs.

7.22.2.74 LightGrey

```
Colour VectSharp.Colours.LightGrey = Colour.FromRgb(211, 211, 211)  [static]
```

LightGrey #D3D3D3

Definition at line 535 of file StandardColours.cs.

7.22.2.75 LightPink

```
Colour VectSharp.Colours.LightPink = Colour.FromRgb(255, 182, 193)  [static]
```

LightPink #FFB6C1

Definition at line 699 of file StandardColours.cs.

7.22.2.76 LightSalmon

```
Colour VectSharp.Colours.LightSalmon = Colour.FromRgb(255, 160, 122)  [static]
```

LightSalmon #FFA07A

Definition at line 691 of file StandardColours.cs.

7.22.2.77 LightSeaGreen

```
Colour VectSharp.Colours.LightSeaGreen = Colour.FromRgb(32, 178, 170)  [static]
```

LightSeaGreen #20B2AA

Definition at line 259 of file StandardColours.cs.

7.22.2.78 LightSkyBlue

```
Colour VectSharp.Colours.LightSkyBlue = Colour.FromRgb(135, 206, 250)  [static]
```

LightSkyBlue #87CEFA

Definition at line 403 of file StandardColours.cs.

7.22.2.79 LightSlateGray

```
Colour VectSharp.Colours.LightSlateGray = Colour.FromRgb(119, 136, 153)  [static]
```

LightSlateGray #778899

Definition at line 355 of file StandardColours.cs.

7.22.2.80 LightSlateGrey

```
Colour VectSharp.Colours.LightSlateGrey = Colour.FromRgb(119, 136, 153)  [static]
```

LightSlateGrey #778899

Definition at line 359 of file StandardColours.cs.

7.22.2.81 LightSteelBlue

```
Colour VectSharp.Colours.LightSteelBlue = Colour.FromRgb(176, 196, 222)  [static]
```

LightSteelBlue #B0C4DE

Definition at line 479 of file StandardColours.cs.

7.22.2.82 LightYellow

```
Colour VectSharp.Colours.LightYellow = Colour.FromRgb(255, 255, 224)  [static]
```

LightYellow #FFFFE0

Definition at line 767 of file StandardColours.cs.

7.22.2.83 Lime

```
Colour VectSharp.Colours.Lime = Colour.FromRgb(0, 255, 0)  [static]
```

Lime #00FF00

Definition at line 235 of file StandardColours.cs.

7.22.2.84 LimeGreen

```
Colour VectSharp.Colours.LimeGreen = Colour.FromRgb(50, 205, 50)  [static]
```

LimeGreen #32CD32

Definition at line 279 of file StandardColours.cs.

7.22.2.85 Linen

```
Colour VectSharp.Colours.Linen = Colour.FromRgb(250, 240, 230)  [static]
```

Linen #FAF0E6

Definition at line 643 of file StandardColours.cs.

7.22.2.86 Magenta

```
Colour VectSharp.Colours.Magenta = Colour.FromRgb(255, 0, 255)  [static]
```

Magenta #FF00FF

Definition at line 663 of file StandardColours.cs.

7.22.2.87 Maroon

```
Colour VectSharp.Colours.Maroon = Colour.FromRgb(128, 0, 0)  [static]
```

Maroon #800000

Definition at line 379 of file StandardColours.cs.

7.22.2.88 MediumAquaMarine

```
Colour VectSharp.Colours.MediumAquaMarine = Colour.FromRgb(102, 205, 170)  [static]
```

MediumAquaMarine #66CDAA

Definition at line 327 of file StandardColours.cs.

7.22.2.89 MediumBlue

```
Colour VectSharp.Colours.MediumBlue = Colour.FromRgb(0, 0, 205)  [static]
```

MediumBlue #0000CD

Definition at line 199 of file StandardColours.cs.

7.22.2.90 MediumOrchid

```
Colour VectSharp.Colours.MediumOrchid = Colour.FromRgb(186, 85, 211)  [static]
```

MediumOrchid #BA55D3

Definition at line 495 of file StandardColours.cs.

7.22.2.91 MediumPurple

```
Colour VectSharp.Colours.MediumPurple = Colour.FromRgb(147, 112, 219)  [static]
```

MediumPurple #9370DB

Definition at line 431 of file StandardColours.cs.

7.22.2.92 MediumSeaGreen

```
Colour VectSharp.Colours.MediumSeaGreen = Colour.FromRgb(60, 179, 113)  [static]
```

MediumSeaGreen #3CB371

Definition at line 283 of file StandardColours.cs.

7.22.2.93 MediumSlateBlue

```
Colour VectSharp.Colours.MediumSlateBlue = Colour.FromRgb(123, 104, 238)  [static]
```

MediumSlateBlue #7B68EE

Definition at line 363 of file StandardColours.cs.

7.22.2.94 MediumSpringGreen

```
Colour VectSharp.Colours.MediumSpringGreen = Colour.FromRgb(0, 250, 154)  [static]
```

MediumSpringGreen #00FA9A

Definition at line 231 of file StandardColours.cs.

7.22.2.95 MediumTurquoise

```
Colour VectSharp.Colours.MediumTurquoise = Colour.FromRgb(72, 209, 204)  [static]
```

MediumTurquoise #48D1CC

Definition at line 303 of file StandardColours.cs.

7.22.2.96 MediumVioletRed

```
Colour VectSharp.Colours.MediumVioletRed = Colour.FromRgb(199, 21, 133)  [static]
```

MediumVioletRed #C71585

Definition at line 511 of file StandardColours.cs.

7.22.2.97 MidnightBlue

```
Colour VectSharp.Colours.MidnightBlue = Colour.FromRgb(25, 25, 112)  [static]
```

MidnightBlue #191970

Definition at line 251 of file StandardColours.cs.

7.22.2.98 MintCream

```
Colour VectSharp.Colours.MintCream = Colour.FromRgb(245, 255, 250)  [static]
```

MintCream #F5FFFA

Definition at line 627 of file StandardColours.cs.

7.22.2.99 MistyRose

```
Colour VectSharp.Colours.MistyRose = Colour.FromRgb(255, 228, 225)  [static]
```

MistyRose #FFE4E1

Definition at line 727 of file StandardColours.cs.

7.22.2.100 Moccasin

```
Colour VectSharp.Colours.Moccasin = Colour.FromRgb(255, 228, 181)  [static]
```

Moccasin #FFE4B5

Definition at line 719 of file StandardColours.cs.

7.22.2.101 NavajoWhite

```
Colour VectSharp.Colours.NavajoWhite = Colour.FromRgb(255, 222, 173)  [static]
```

NavajoWhite #FFDEAD

Definition at line 715 of file StandardColours.cs.

7.22.2.102 Navy

```
Colour VectSharp.Colours.Navy = Colour.FromRgb(0, 0, 128)  [static]
```

Navy #000080

Definition at line 191 of file StandardColours.cs.

7.22.2.103 OldLace

```
Colour VectSharp.Colours.OldLace = Colour.FromRgb(253, 245, 230)  [static]
```

OldLace #FDF5E6

Definition at line 651 of file StandardColours.cs.

7.22.2.104 Olive

```
Colour VectSharp.Colours.Olive = Colour.FromRgb(128, 128, 0)  [static]
```

Olive #808000

Definition at line 387 of file StandardColours.cs.

7.22.2.105 OliveDrab

```
Colour VectSharp.Colours.OliveDrab = Colour.FromRgb(107, 142, 35)  [static]
```

OliveDrab #6B8E23

Definition at line 343 of file StandardColours.cs.

7.22.2.106 Orange

```
Colour VectSharp.Colours.Orange = Colour.FromRgb(255, 165, 0)  [static]
```

Orange #FFA500

Definition at line 695 of file StandardColours.cs.

7.22.2.107 OrangeRed

```
Colour VectSharp.Colours.OrangeRed = Colour.FromRgb(255, 69, 0)  [static]
```

OrangeRed #FF4500

Definition at line 671 of file StandardColours.cs.

7.22.2.108 Orchid

```
Colour VectSharp.Colours.Orchid = Colour.FromRgb(218, 112, 214)  [static]
```

Orchid #DA70D6

Definition at line 543 of file StandardColours.cs.

7.22.2.109 PaleGoldenRod

```
Colour VectSharp.Colours.PaleGoldenRod = Colour.FromRgb(238, 232, 170)  [static]
```

PaleGoldenRod #EEE8AA

Definition at line 587 of file StandardColours.cs.

7.22.2.110 PaleGreen

```
Colour VectSharp.Colours.PaleGreen = Colour.FromRgb(152, 251, 152)  [static]
```

PaleGreen #98FB98

Definition at line 439 of file StandardColours.cs.

7.22.2.111 PaleTurquoise

```
Colour VectSharp.Colours.PaleTurquoise = Colour.FromRgb(175, 238, 238)  [static]
```

PaleTurquoise #AFEEEE

Definition at line 475 of file StandardColours.cs.

7.22.2.112 PaleVioletRed

```
Colour VectSharp.Colours.PaleVioletRed = Colour.FromRgb(219, 112, 147)  [static]
```

PaleVioletRed #DB7093

Definition at line 551 of file StandardColours.cs.

7.22.2.113 PapayaWhip

```
Colour VectSharp.Colours.PapayaWhip = Colour.FromRgb(255, 239, 213)  [static]
```

PapayaWhip #FFEFD5

Definition at line 735 of file StandardColours.cs.

7.22.2.114 PeachPuff

```
Colour VectSharp.Colours.PeachPuff = Colour.FromRgb(255, 218, 185)  [static]
```

PeachPuff #FFDAB9

Definition at line 711 of file StandardColours.cs.

7.22.2.115 Peru

```
Colour VectSharp.Colours.Peru = Colour.FromRgb(205, 133, 63)  [static]
```

Peru #CD853F

Definition at line 519 of file StandardColours.cs.

7.22.2.116 Pink

```
Colour VectSharp.Colours.Pink = Colour.FromRgb(255, 192, 203)  [static]
```

Pink #FFC0CB

Definition at line 703 of file StandardColours.cs.

7.22.2.117 Plum

```
Colour VectSharp.Colours.Plum = Colour.FromRgb(221, 160, 221)  [static]
```

Plum #DDA0DD

Definition at line 563 of file StandardColours.cs.

7.22.2.118 PowderBlue

```
Colour VectSharp.Colours.PowderBlue = Colour.FromRgb(176, 224, 230)  [static]
```

PowderBlue #B0E0E6

Definition at line 483 of file StandardColours.cs.

7.22.2.119 Purple

```
Colour VectSharp.Colours.Purple = Colour.FromRgb(128, 0, 128)  [static]
```

Purple #800080

Definition at line 383 of file StandardColours.cs.

7.22.2.120 RebeccaPurple

```
Colour VectSharp.Colours.RebeccaPurple = Colour.FromRgb(102, 51, 153)  [static]
```

RebeccaPurple #663399

Definition at line 323 of file StandardColours.cs.

7.22.2.121 Red

```
Colour VectSharp.Colours.Red = Colour.FromRgb(255, 0, 0)  [static]
```

Red #FF0000

Definition at line 655 of file StandardColours.cs.

7.22.2.122 RosyBrown

```
Colour VectSharp.Colours.RosyBrown = Colour.FromRgb(188, 143, 143)  [static]
```

RosyBrown #BC8F8F

Definition at line 499 of file StandardColours.cs.

7.22.2.123 RoyalBlue

```
Colour VectSharp.Colours.RoyalBlue = Colour.FromRgb(65, 105, 225)  [static]
```

RoyalBlue #4169E1

Definition at line 291 of file StandardColours.cs.

7.22.2.124 SaddleBrown

```
Colour VectSharp.Colours.SaddleBrown = Colour.FromRgb(139, 69, 19)  [static]
```

SaddleBrown #8B4513

Definition at line 419 of file StandardColours.cs.

7.22.2.125 Salmon

```
Colour VectSharp.Colours.Salmon = Colour.FromRgb(250, 128, 114)  [static]
```

Salmon #FA8072

Definition at line 635 of file StandardColours.cs.

7.22.2.126 SandyBrown

```
Colour VectSharp.Colours.SandyBrown = Colour.FromRgb(244, 164, 96)  [static]
```

SandyBrown #F4A460

Definition at line 611 of file StandardColours.cs.

7.22.2.127 SeaGreen

```
Colour VectSharp.Colours.SeaGreen = Colour.FromRgb(46, 139, 87)  [static]
```

SeaGreen #2E8B57

Definition at line 267 of file StandardColours.cs.

7.22.2.128 SeaShell

```
Colour VectSharp.Colours.SeaShell = Colour.FromRgb(255, 245, 238)  [static]
```

SeaShell #FFF5EE

Definition at line 743 of file StandardColours.cs.

7.22.2.129 Sienna

```
Colour VectSharp.Colours.Sienna = Colour.FromRgb(160, 82, 45)  [static]
```

Sienna #A0522D

Definition at line 451 of file StandardColours.cs.

7.22.2.130 Silver

```
Colour VectSharp.Colours.Silver = Colour.FromRgb(192, 192, 192)  [static]
```

Silver #C0C0C0

Definition at line 507 of file StandardColours.cs.

7.22.2.131 SkyBlue

```
Colour VectSharp.Colours.SkyBlue = Colour.FromRgb(135, 206, 235)  [static]
```

SkyBlue #87CEEB

Definition at line 399 of file StandardColours.cs.

7.22.2.132 SlateBlue

```
Colour VectSharp.Colours.SlateBlue = Colour.FromRgb(106, 90, 205)  [static]
```

SlateBlue #6A5ACD

Definition at line 339 of file StandardColours.cs.

7.22.2.133 SlateGray

```
Colour VectSharp.Colours.SlateGray = Colour.FromRgb(112, 128, 144)  [static]
```

SlateGray #708090

Definition at line 347 of file StandardColours.cs.

7.22.2.134 SlateGrey

```
Colour VectSharp.Colours.SlateGrey = Colour.FromRgb(112, 128, 144)  [static]
```

SlateGrey #708090

Definition at line 351 of file StandardColours.cs.

7.22.2.135 Snow

```
Colour VectSharp.Colours.Snow = Colour.FromRgb(255, 250, 250)  [static]
```

Snow #FFFAFA

Definition at line 759 of file StandardColours.cs.

7.22.2.136 SpringGreen

```
Colour VectSharp.Colours.SpringGreen = Colour.FromRgb(0, 255, 127)  [static]
```

SpringGreen #00FF7F

Definition at line 239 of file StandardColours.cs.

7.22.2.137 SteelBlue

```
Colour VectSharp.Colours.SteelBlue = Colour.FromRgb(70, 130, 180)  [static]
```

SteelBlue #4682B4

Definition at line 295 of file StandardColours.cs.

7.22.2.138 Tan

```
Colour VectSharp.Colours.Tan = Colour.FromRgb(210, 180, 140)  [static]
```

Tan #D2B48C

Definition at line 527 of file StandardColours.cs.

7.22.2.139 Teal

```
Colour VectSharp.Colours.Teal = Colour.FromRgb(0, 128, 128)  [static]
```

Teal #008080

Definition at line 215 of file StandardColours.cs.

7.22.2.140 Thistle

```
Colour VectSharp.Colours.Thistle = Colour.FromRgb(216, 191, 216)  [static]
```

Thistle #D8BFD8

Definition at line 539 of file StandardColours.cs.

7.22.2.141 Tomato

```
Colour VectSharp.Colours.Tomato = Colour.FromRgb(255, 99, 71)  [static]
```

Tomato #FF6347

Definition at line 675 of file StandardColours.cs.

7.22.2.142 Turquoise

```
Colour VectSharp.Colours.Turquoise = Colour.FromRgb(64, 224, 208)  [static]
```

Turquoise #40E0D0

Definition at line 287 of file StandardColours.cs.

7.22.2.143 Violet

```
Colour VectSharp.Colours.Violet = Colour.FromRgb(238, 130, 238)  [static]
```

Violet #EE82EE

Definition at line 583 of file StandardColours.cs.

7.22.2.144 Wheat

```
Colour VectSharp.Colours.Wheat = Colour.FromRgb(245, 222, 179)  [static]
```

Wheat #F5DEB3

Definition at line 615 of file StandardColours.cs.

7.22.2.145 White

```
Colour VectSharp.Colours.White = Colour.FromRgb(255, 255, 255)  [static]
```

White #FFFFFF

Definition at line 775 of file StandardColours.cs.

7.22.2.146 WhiteSmoke

```
Colour VectSharp.Colours.WhiteSmoke = Colour.FromRgb(245, 245, 245)  [static]
```

WhiteSmoke #F5F5F5

Definition at line 623 of file StandardColours.cs.

7.22.2.147 Yellow

```
Colour VectSharp.Colours.Yellow = Colour.FromRgb(255, 255, 0)  [static]
```

Yellow #FFFF00

Definition at line 763 of file StandardColours.cs.

7.22.2.148 YellowGreen

```
Colour VectSharp.Colours.YellowGreen = Colour.FromRgb(154, 205, 50)  [static]
```

YellowGreen #9ACD32

Definition at line 447 of file StandardColours.cs.

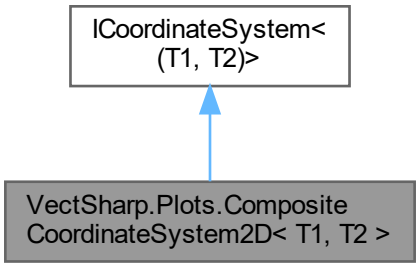
The documentation for this class was generated from the following file:

- VectSharp/StandardColours.cs

7.23 VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 > Class Template Reference

Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.

Inheritance diagram for VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >:



Public Member Functions

- CompositeCoordinateSystem2D (ICoordinateSystem1D< T1 > coordinateSystemX, ICoordinateSystem1D< T2 > coordinateSystemY)

  Creates a new CompositeCoordinateSystem2D<T1, T2> from two 1-D coordinate system.
- Point ToPlotCoordinates ((T1, T2) dataPoint)

Properties

- ICoordinateSystem1D< T1 > CoordinateSystemX `[get, set]`

  The first coordinate system.
- ICoordinateSystem1D< T2 > CoordinateSystemY `[get, set]`

  The second coordinate system.

7.23.1 Detailed Description

Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.

Template Parameters

T1	The type for the first coordinate system.
T2	The type for the second coordinate system.

Definition at line 1136 of file CoordinateSystems.cs.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 CompositeCoordinateSystem2D()

```
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.CompositeCoordinateSystem2D (
            ICoordinateSystem1D< T1 > coordinateSystemX,
            ICoordinateSystem1D< T2 > coordinateSystemY )
```

Creates a new CompositeCoordinateSystem2D<T1, T2> from two 1-D coordinate system.

Parameters

coordinateSystemX	The first coordinate system.
coordinateSystemY	The second coordinate system.

Definition at line 1153 of file CoordinateSystems.cs.

7.23.3 Member Function Documentation

7.23.3.1 ToPlotCoordinates()

```
Point VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.ToPlotCoordinates (
            (T1, T2) dataPoint )
```

Definition at line 1160 of file CoordinateSystems.cs.

7.23.4 Property Documentation

7.23.4.1 CoordinateSystemX

```
ICoordinateSystem1D<T1> VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.Coordinate↩
SystemX [get], [set]
```

The first coordinate system.

Definition at line 1141 of file CoordinateSystems.cs.

7.23.4.2 CoordinateSystemY

```
ICoordinateSystem1D<T2> VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.Coordinate↩
SystemY [get], [set]
```

The second coordinate system.

Definition at line 1146 of file CoordinateSystems.cs.

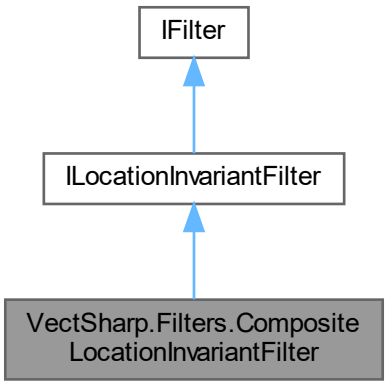
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.24 VectSharp.Filters.CompositeLocationInvariantFilter Class Reference

Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.

Inheritance diagram for VectSharp.Filters.CompositeLocationInvariantFilter:



Public Member Functions

- CompositeLocationInvariantFilter (IEnumerable< ILocationInvariantFilter > filters)

  Creates a new CompositeLocationInvariantFilter with the specified filters.
- CompositeLocationInvariantFilter (params ILocationInvariantFilter[ ] filters)

  Creates a new CompositeLocationInvariantFilter with the specified filters.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- ImmutableList< ILocationInvariantFilter > Filters `[get]`

  The filters that are applied by this filter.

7.24.1 Detailed Description

Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.

Definition at line 26 of file CompositeFilter.cs.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 CompositeLocationInvariantFilter() [1/2]

```
VectSharp.Filters.CompositeLocationInvariantFilter.CompositeLocationInvariantFilter (
            IEnumerable< ILocationInvariantFilter > filters )
```

Creates a new CompositeLocationInvariantFilter with the specified filters.

Parameters

filters	The filters that will be applied by the new filter.

Definition at line 43 of file CompositeFilter.cs.

7.24.2.2 CompositeLocationInvariantFilter() [2/2]

```
VectSharp.Filters.CompositeLocationInvariantFilter.CompositeLocationInvariantFilter (
            params ILocationInvariantFilter[] filters )
```

Creates a new CompositeLocationInvariantFilter with the specified filters.

Parameters

filters	The filters that will be applied by the new filter.

Definition at line 70 of file CompositeFilter.cs.

7.24.3 Member Function Documentation

7.24.3.1 Filter()

```
RasterImage VectSharp.Filters.CompositeLocationInvariantFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 91 of file CompositeFilter.cs.

7.24.4 Property Documentation

7.24.4.1 BottomRightMargin

```
Point VectSharp.Filters.CompositeLocationInvariantFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 32 of file CompositeFilter.cs.

7.24.4.2 Filters

```
ImmutableList<ILocationInvariantFilter> VectSharp.Filters.CompositeLocationInvariantFilter.↩
Filters  [get]
```

The filters that are applied by this filter.

Definition at line 37 of file CompositeFilter.cs.

7.24.4.3 TopLeftMargin

```
Point VectSharp.Filters.CompositeLocationInvariantFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 29 of file CompositeFilter.cs.

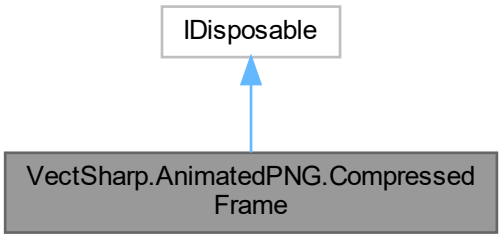
The documentation for this class was generated from the following file:

- VectSharp/Filters/CompositeFilter.cs

7.25 VectSharp.AnimatedPNG.CompressedFrame Class Reference

Represents an individual frame of a PNG animation.

Inheritance diagram for VectSharp.AnimatedPNG.CompressedFrame:



Public Member Functions

- unsafe CompressedFrame (DisposableIntPtr rawFrameData, int width, int height, bool hasAlpha, double duration)

  Creates a new CompressedFrame from raw pixel data in RGB or RGBA format.
- unsafe CompressedFrame (DisposableIntPtr rawFrameData, DisposableIntPtr otherFrameData, bool isFirstFrame, int width, int height, bool hasAlpha, double duration)

  Creates a new CompressedFrame from raw pixel data in RGB or RGBA format, applying inter-frame compression based on another frame.
- void Dispose ()

Properties

- double Duration `[get]`

  The duration of the frame in milliseconds.

7.25.1 Detailed Description

Represents an individual frame of a PNG animation.

Definition at line 56 of file ImageFormats.cs.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 CompressedFrame() [1/2]

```
unsafe VectSharp.AnimatedPNG.CompressedFrame.CompressedFrame (
            DisposableIntPtr rawFrameData,
            int width,
            int height,
            bool hasAlpha,
            double duration )
```

Creates a new CompressedFrame from raw pixel data in RGB or RGBA format.

Parameters

rawFrameData	A pointer to the raw pixel data for the frame.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the image data is in RGBA format, set this to `true`; if it is in RGB format, set it to `false`.
duration	The duration of the frame in milliseconds.

Definition at line 76 of file ImageFormats.cs.

7.25.2.2 CompressedFrame() [2/2]

```
unsafe VectSharp.AnimatedPNG.CompressedFrame.CompressedFrame (
            DisposableIntPtr rawFrameData,
            DisposableIntPtr otherFrameData,
            bool isFirstFrame,
            int width,
            int height,
            bool hasAlpha,
            double duration )
```

Creates a new CompressedFrame from raw pixel data in RGB or RGBA format, applying inter-frame compression based on another frame.

Parameters

rawFrameData	A pointer to the raw pixel data for the new frame.
otherFrameData	A pointer to the raw pixel data for the frame to use as a reference. This can either be the first frame in the animation, or the frame immediately preceeding the current frame.
isFirstFrame	If *otherFrameData* is the raw pixel data for the first frame in the animation, set this to `true`; if it is the immediately preceeding frame, set this to `false`.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the image data is in RGBA format, set this to `true`; if it is in RGB format, set it to `false`.
duration	The duration of the frame in milliseconds.

Definition at line 93 of file ImageFormats.cs.

7.25.3 Member Function Documentation

7.25.3.1 Dispose()

```
void VectSharp.AnimatedPNG.CompressedFrame.Dispose ( )
```

Definition at line 165 of file ImageFormats.cs.

7.25.4 Property Documentation

7.25.4.1 Duration

```
double VectSharp.AnimatedPNG.CompressedFrame.Duration  [get]
```

The duration of the frame in milliseconds.

Definition at line 64 of file ImageFormats.cs.

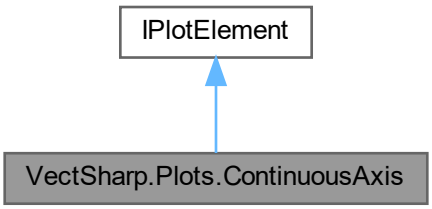
The documentation for this class was generated from the following file:

- VectSharp/ImageFormats.cs

7.26 VectSharp.Plots.ContinuousAxis Class Reference

A plot element that draws an axis line and an arrow.

Inheritance diagram for VectSharp.Plots.ContinuousAxis:



Public Member Functions

- ContinuousAxis (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Creates a new ContinuousAxis instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.
- double ArrowSize = 3 `[get, set]`

  The size of the arrow at the end of the axis, expressed in plot space coordinates.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the axis.
- string Tag `[get, set]`

  A tag to identify the axis in the plot.

7.26.1 Detailed Description

A plot element that draws an axis line and an arrow.

Definition at line 26 of file Axes.cs.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 ContinuousAxis()

```
VectSharp.Plots.ContinuousAxis.ContinuousAxis (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Creates a new ContinuousAxis instance.

Parameters

startPoint	The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.
endPoint	The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 65 of file Axes.cs.

7.26.3 Member Function Documentation

7.26.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxis.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 73 of file Axes.cs.

7.26.4 Property Documentation

7.26.4.1 ArrowSize

```
double VectSharp.Plots.ContinuousAxis.ArrowSize = 3  [get], [set]
```

The size of the arrow at the end of the axis, expressed in plot space coordinates.

Definition at line 41 of file Axes.cs.

7.26.4.2 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxis.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 46 of file Axes.cs.

7.26.4.3 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxis.EndPoint  [get], [set]
```

The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.

Definition at line 36 of file Axes.cs.

7.26.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxis.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the axis.

Definition at line 52 of file Axes.cs.

7.26.4.5 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxis.StartPoint  [get], [set]
```

The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.

Definition at line 31 of file Axes.cs.

7.26.4.6 Tag

```
string VectSharp.Plots.ContinuousAxis.Tag  [get], [set]
```

A tag to identify the axis in the plot.

Definition at line 57 of file Axes.cs.

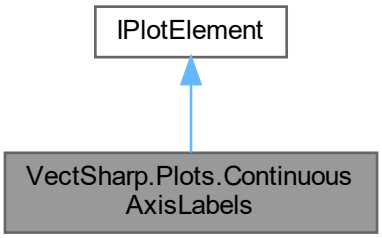
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.27 VectSharp.Plots.ContinuousAxisLabels Class Reference

A plot element that draws equally spaced labels on an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisLabels:



Public Member Functions

- ContinuousAxisLabels (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Create a new ContinuousAxisLabels instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis, expressed in data space coordinates.
- double IntervalCount = 10 `[get, set]`

  The number of intervals between labels. Note that the number of labels will be one greater than this.
- Func< int, double > Position = _ => 10 `[get, set]`

  The distance of the label anchor from the point on the axis. This should be set to a function accepting an `int` argument representing the index of the label, and return a `double` representing the distance of the label from the axis, in plot space coordinates.
- Func< IReadOnlyList< double >, int, IEnumerable< FormattedText > > TextFormat `[get, set]`

  A function used to determine what text to draw at each label. You should set this to a function accepting two parameters: an IReadOnlyList<T> of `double`s, representing the coordinates of the point the label refers to, and an `int` representing the index of the label. The function should return an IEnumerable<T> of FormattedText objects, representing the text that will be drawn.
- double? Rotation = null `[get, set]`

  The orientation of the label with respect to the horizontal. If this is `null`, the labels will be perpendicular to the axis.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the label anchor.
- TextAnchors Alignment = TextAnchors.Left `[get, set]`

  The alignment for the label anchor.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the labels.
- string Tag `[get, set]`

  A tag to identify the labels in the plot.

7.27.1 Detailed Description

A plot element that draws equally spaced labels on an axis.

Definition at line 302 of file Axes.cs.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 ContinuousAxisLabels()

```
VectSharp.Plots.ContinuousAxisLabels.ContinuousAxisLabels (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ContinuousAxisLabels instance.

Parameters

startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 447 of file Axes.cs.

7.27.3 Member Function Documentation

7.27.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisLabels.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 456 of file Axes.cs.

7.27.4 Property Documentation

7.27.4.1 Alignment

```
TextAnchors VectSharp.Plots.ContinuousAxisLabels.Alignment = TextAnchors.Left  [get], [set]
```

The alignment for the label anchor.

Definition at line 423 of file Axes.cs.

7.27.4.2 Baseline

```
TextBaselines VectSharp.Plots.ContinuousAxisLabels.Baseline = TextBaselines.Middle  [get],
[set]
```

The baseline for the label anchor.

Definition at line 418 of file Axes.cs.

7.27.4.3 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisLabels.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 428 of file Axes.cs.

7.27.4.4 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisLabels.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 312 of file Axes.cs.

7.27.4.5 IntervalCount

```
double VectSharp.Plots.ContinuousAxisLabels.IntervalCount = 10  [get], [set]
```

The number of intervals between labels. Note that the number of labels will be one greater than this.

Definition at line 317 of file Axes.cs.

7.27.4.6 Position

```
Func<int, double> VectSharp.Plots.ContinuousAxisLabels.Position = _ => 10  [get], [set]
```

The distance of the label anchor from the point on the axis. This should be set to a function accepting an `int` argument representing the index of the label, and return a `double` representing the distance of the label from the axis, in plot space coordinates.

Definition at line 324 of file Axes.cs.

7.27.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisLabels.PresentationAttributes
= new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the labels.

Definition at line 434 of file Axes.cs.

7.27.4.8 Rotation

```
double?  VectSharp.Plots.ContinuousAxisLabels.Rotation = null  [get], [set]
```

The orientation of the label with respect to the horizontal. If this is `null`, the labels will be perpendicular to the axis.

Definition at line 339 of file Axes.cs.

7.27.4.9 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisLabels.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 307 of file Axes.cs.

7.27.4.10 Tag

```
string VectSharp.Plots.ContinuousAxisLabels.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 439 of file Axes.cs.

7.27.4.11 TextFormat

```
Func<IReadOnlyList<double>, int, IEnumerable<FormattedText> > VectSharp.Plots.Continuous↩
AxisLabels.TextFormat  [get], [set]
```

A function used to determine what text to draw at each label. You should set this to a function accepting two parameters: an IReadOnlyList<T> of `double`s, representing the coordinates of the point the label refers to, and an `int` representing the index of the label. The function should return an IEnumerable<T> of FormattedText objects, representing the text that will be drawn.

Definition at line 333 of file Axes.cs.

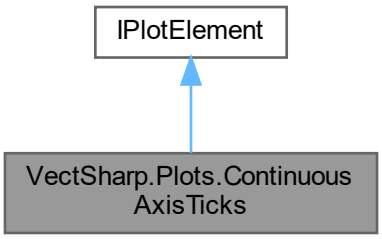
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.28 VectSharp.Plots.ContinuousAxisTicks Class Reference

A plot element that draws equally spaced ticks on an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisTicks:



Public Member Functions

- ContinuousAxisTicks (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Creates a new ContinuousAxisTicks instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis, expressed in data space coordinates.
- double IntervalCount = 10 `[get, set]`

  The number of intervals between ticks. Note that the number of ticks will be one greater than this.
- Func< int, double > SizeAbove = i => i % 2 == 0 ? 3 : 2 `[get, set]`

  The size of the ticks "above" the axis. What "above" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.
- Func< int, double > SizeBelow = i => i % 2 == 0 ? 3 : 2 `[get, set]`

  The size of the ticks "below" the axis. What "below" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

The coordinate system used to transform the points from data space to plot space.

- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the ticks.

- string Tag `[get, set]`

  A tag to identify the ticks in the plot.

7.28.1 Detailed Description

A plot element that draws equally spaced ticks on an axis.

Definition at line 167 of file Axes.cs.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 ContinuousAxisTicks()

```
VectSharp.Plots.ContinuousAxisTicks.ContinuousAxisTicks (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Creates a new ContinuousAxisTicks instance.

Parameters

startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 220 of file Axes.cs.

7.28.3 Member Function Documentation

7.28.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisTicks.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 228 of file Axes.cs.

7.28.4 Property Documentation

7.28.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisTicks.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 201 of file Axes.cs.

7.28.4.2 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTicks.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 177 of file Axes.cs.

7.28.4.3 IntervalCount

```
double VectSharp.Plots.ContinuousAxisTicks.IntervalCount = 10  [get], [set]
```

The number of intervals between ticks. Note that the number of ticks will be one greater than this.

Definition at line 182 of file Axes.cs.

7.28.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisTicks.PresentationAttributes =
new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the ticks.

Definition at line 207 of file Axes.cs.

7.28.4.5 SizeAbove

```
Func<int, double> VectSharp.Plots.ContinuousAxisTicks.SizeAbove = i => i % 2 == 0 ?  3 :  2
[get], [set]
```

The size of the ticks "above" the axis. What "above" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.

Definition at line 189 of file Axes.cs.

7.28.4.6 SizeBelow

```
Func<int, double> VectSharp.Plots.ContinuousAxisTicks.SizeBelow = i => i % 2 == 0 ?  3 :  2
[get], [set]
```

The size of the ticks "below" the axis. What "below" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.

Definition at line 196 of file Axes.cs.

7.28.4.7 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTicks.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 172 of file Axes.cs.

7.28.4.8 Tag

```
string VectSharp.Plots.ContinuousAxisTicks.Tag  [get], [set]
```

A tag to identify the ticks in the plot.

Definition at line 212 of file Axes.cs.

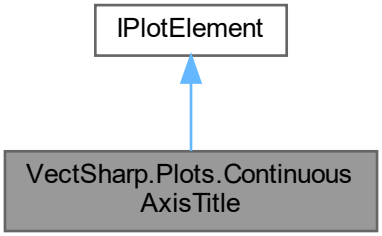
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.29 VectSharp.Plots.ContinuousAxisTitle Class Reference

A plot element that draws a title for an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisTitle:



Public Member Functions

- ContinuousAxisTitle (IEnumerable< FormattedText > title, IReadOnlyList< double > startPoint, IRead↩ OnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Create a new ContinuousAxisTitle instance.
- ContinuousAxisTitle (string title, IReadOnlyList< double > startPoint, IReadOnlyList< double > end↩ Point, IContinuousCoordinateSystem coordinateSystem, PlotElementPresentationAttributes presentation↩ Attributes=null)

  Create a new ContinuousAxisTitle instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis, expressed in data space coordinates.
- double Position = 30 `[get, set]`

  The distance between the title and the axis, in plot space coordinates.
- IEnumerable< FormattedText > Title `[get, set]`

  The axis title to draw.
- bool FollowAxis = true `[get, set]`

  If the axis is not a straight line (e.g., because the coordinate system is not linear), if this is `true` the title will follow the shape of the axis. If this is `false`, the title will always be drawn on a straight line.
- double? Rotation = null `[get, set]`

  Orientation of the title with respect to the horizontal. If this is `null`, the title is parallel to the axis.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the title anchor.
- TextAnchors Alignment = TextAnchors.Left `[get, set]`

  The alignment for the title anchor.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke = null, Fill = Colours.Black } `[get, set]`

  Presentation attributes determining the appearance of the title.
- string Tag `[get, set]`

  A tag to identify the labels in the plot.

7.29.1 Detailed Description

A plot element that draws a title for an axis.

Definition at line 556 of file Axes.cs.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 ContinuousAxisTitle() [1/2]

```
VectSharp.Plots.ContinuousAxisTitle.ContinuousAxisTitle (
            IEnumerable< FormattedText > title,
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ContinuousAxisTitle instance.

Parameters

title	The title to draw on the axis.
startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 624 of file Axes.cs.

7.29.2.2 ContinuousAxisTitle() [2/2]

```
VectSharp.Plots.ContinuousAxisTitle.ContinuousAxisTitle (
            string title,
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem,
            PlotElementPresentationAttributes presentationAttributes = null )
```

Create a new ContinuousAxisTitle instance.

Parameters

title	The title to draw on the axis.
startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
presentationAttributes	Presentation attributes determining the appearance of the title.

Definition at line 640 of file Axes.cs.

7.29.3 Member Function Documentation

7.29.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisTitle.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 671 of file Axes.cs.

7.29.4 Property Documentation

7.29.4.1 Alignment

```
TextAnchors VectSharp.Plots.ContinuousAxisTitle.Alignment = TextAnchors.Left  [get], [set]
```

The alignment for the title anchor.

Definition at line 599 of file Axes.cs.

7.29.4.2 Baseline

```
TextBaselines VectSharp.Plots.ContinuousAxisTitle.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the title anchor.

Definition at line 594 of file Axes.cs.

7.29.4.3 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisTitle.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 604 of file Axes.cs.

7.29.4.4 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTitle.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 566 of file Axes.cs.

7.29.4.5 FollowAxis

```
bool VectSharp.Plots.ContinuousAxisTitle.FollowAxis = true  [get], [set]
```

If the axis is not a straight line (e.g., because the coordinate system is not linear), if this is `true` the title will follow the shape of the axis. If this is `false`, the title will always be drawn on a straight line.

Definition at line 583 of file Axes.cs.

7.29.4.6 Position

```
double VectSharp.Plots.ContinuousAxisTitle.Position = 30  [get], [set]
```

The distance between the title and the axis, in plot space coordinates.

Definition at line 571 of file Axes.cs.

7.29.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisTitle.PresentationAttributes =
new PlotElementPresentationAttributes() { Font = new Font(FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBold), 14), Stroke = null, Fill = Colours.Black }  [get],
[set]
```

Presentation attributes determining the appearance of the title.

Definition at line 610 of file Axes.cs.

7.29.4.8 Rotation

```
double?  VectSharp.Plots.ContinuousAxisTitle.Rotation = null  [get], [set]
```

Orientation of the title with respect to the horizontal. If this is `null`, the title is parallel to the axis.

Definition at line 589 of file Axes.cs.

7.29.4.9 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTitle.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 561 of file Axes.cs.

7.29.4.10 Tag

```
string VectSharp.Plots.ContinuousAxisTitle.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 615 of file Axes.cs.

7.29.4.11 Title

```
IEnumerable<FormattedText> VectSharp.Plots.ContinuousAxisTitle.Title  [get], [set]
```

The axis title to draw.

Definition at line 576 of file Axes.cs.

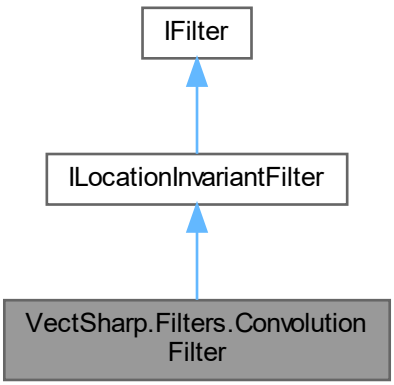
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.30 VectSharp.Filters.ConvolutionFilter Class Reference

Represents a filter that applies a matrix convolution to the image.

Inheritance diagram for VectSharp.Filters.ConvolutionFilter:



Public Member Functions

- ConvolutionFilter (double[,] kernel, double scale, bool preserveAlpha=true, double normalisation=1, double bias=0)

  Creates a new ConvolutionFilter with the specified parameters.
- virtual RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- virtual double[,] Kernel `[get]`

  The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.
- virtual double Normalisation = 1 `[get]`

  The normalisation value that is applies to the kernel.
- virtual double Bias = 0 `[get]`

  The bias value that is added to every colour component when the filter is applied.
- virtual double Scale `[get]`

  The scale relating the size of the kernel to graphics units.
- virtual bool PreserveAlpha = true `[get]`

  If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.

7.30.1 Detailed Description

Represents a filter that applies a matrix convolution to the image.

Definition at line 26 of file ConvolutionFilter.cs.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 ConvolutionFilter()

```
VectSharp.Filters.ConvolutionFilter.ConvolutionFilter (
            double kernel[,],
            double scale,
            bool preserveAlpha = true,
            double normalisation = 1,
            double bias = 0 )
```

Creates a new ConvolutionFilter with the specified parameters.

Parameters

kernel	The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.
scale	The scale relating the size of the kernel to graphics units.
preserveAlpha	If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.
normalisation	The normalisation value that is applies to the kernel.
bias	The bias value that is added to every colour component when the filter is applied.

Exceptions

ArgumentException	This exception is thrown when the kernel dimensions are not odd numbers.

Definition at line 67 of file ConvolutionFilter.cs.

7.30.3 Member Function Documentation

7.30.3.1 Filter()

```
virtual RasterImage VectSharp.Filters.ConvolutionFilter.Filter (
            RasterImage image,
            double scale )  [virtual]
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 88 of file ConvolutionFilter.cs.

7.30.4 Property Documentation

7.30.4.1 Bias

```
virtual double VectSharp.Filters.ConvolutionFilter.Bias = 0  [get]
```

The bias value that is added to every colour component when the filter is applied.

Definition at line 46 of file ConvolutionFilter.cs.

7.30.4.2 BottomRightMargin

```
Point VectSharp.Filters.ConvolutionFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 31 of file ConvolutionFilter.cs.

7.30.4.3 Kernel

```
virtual double [,] VectSharp.Filters.ConvolutionFilter.Kernel  [get]
```

The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.

Definition at line 36 of file ConvolutionFilter.cs.

7.30.4.4 Normalisation

```
virtual double VectSharp.Filters.ConvolutionFilter.Normalisation = 1  [get]
```

The normalisation value that is applies to the kernel.

Definition at line 41 of file ConvolutionFilter.cs.

7.30.4.5 PreserveAlpha

```
virtual bool VectSharp.Filters.ConvolutionFilter.PreserveAlpha = true  [get]
```

If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.

Definition at line 56 of file ConvolutionFilter.cs.

7.30.4.6 Scale

```
virtual double VectSharp.Filters.ConvolutionFilter.Scale  [get]
```

The scale relating the size of the kernel to graphics units.

Definition at line 51 of file ConvolutionFilter.cs.

7.30.4.7 TopLeftMargin

```
Point VectSharp.Filters.ConvolutionFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 29 of file ConvolutionFilter.cs.

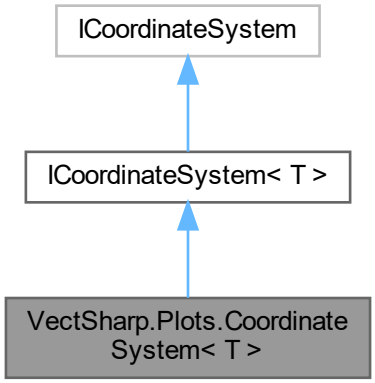
The documentation for this class was generated from the following file:

- VectSharp/Filters/ConvolutionFilter.cs

7.31 VectSharp.Plots.CoordinateSystem< T > Class Template Reference

A coordinate system using a custom method to transform data points.

Inheritance diagram for VectSharp.Plots.CoordinateSystem< T >:



Public Member Functions

- CoordinateSystem (Func< T, Point > coordinateFunction)

  Create a new CoordinateSystem< T > using the specified method.
- Point ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a plot Point.

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A Point representing the dataPoint in plot space.

Properties

- Func< T, Point > CoordinateFunction `[get, set]`

  The method used to transform the data elements into plot Points.

7.31.1 Detailed Description

A coordinate system using a custom method to transform data points.

Template Parameters

T	The type of data elements.

Definition at line 63 of file CoordinateSystems.cs.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 CoordinateSystem()

```
VectSharp.Plots.CoordinateSystem< T >.CoordinateSystem (
            Func< T, Point > coordinateFunction )
```

Create a new CoordinateSystem<T> using the specified method.

Parameters

coordinateFunction	The method used to transform the data elements into plot Points.

Definition at line 74 of file CoordinateSystems.cs.

7.31.3 Member Function Documentation

7.31.3.1 ToPlotCoordinates()

```
Point VectSharp.Plots.CoordinateSystem< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a plot Point.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem< T >.

Definition at line 80 of file CoordinateSystems.cs.

7.31.4 Property Documentation

7.31.4.1 CoordinateFunction

```
Func<T, Point> VectSharp.Plots.CoordinateSystem< T >.CoordinateFunction  [get], [set]
```

The method used to transform the data elements into plot Points.

Definition at line 68 of file CoordinateSystems.cs.

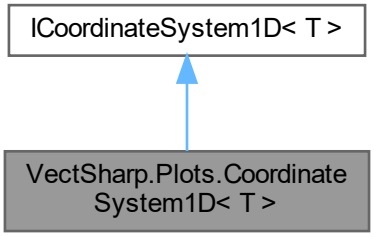
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.32 VectSharp.Plots.CoordinateSystem1D< T > Class Template Reference

A coordinate system using a custom method to transform data points.

Inheritance diagram for VectSharp.Plots.CoordinateSystem1D< T >:



Public Member Functions

- CoordinateSystem1D (Func< T, double > coordinateFunction)

  Creates a new CoordinateSystem1D<T> using the specified coordinateFunction .
- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`.

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A `double` *representing the dataPoint in plot space.*

Properties

- Func< T, double > CoordinateFunction `[get, set]`

  The method used to transform data points.

7.32.1 Detailed Description

A coordinate system using a custom method to transform data points.

Template Parameters

T	The type of data elements.

Definition at line 1055 of file CoordinateSystems.cs.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 CoordinateSystem1D()

```
VectSharp.Plots.CoordinateSystem1D< T >.CoordinateSystem1D (
            Func< T, double > coordinateFunction )
```

Creates a new CoordinateSystem1D<T> using the specified *coordinateFunction* .

Parameters

coordinateFunction	The method used to transform data points.

Definition at line 1066 of file CoordinateSystems.cs.

7.32.3 Member Function Documentation

7.32.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.CoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem1D< T >.

Definition at line 1072 of file CoordinateSystems.cs.

7.32.4 Property Documentation

7.32.4.1 CoordinateFunction

```
Func<T, double> VectSharp.Plots.CoordinateSystem1D< T >.CoordinateFunction  [get], [set]
```

The method used to transform data points.

Definition at line 1060 of file CoordinateSystems.cs.

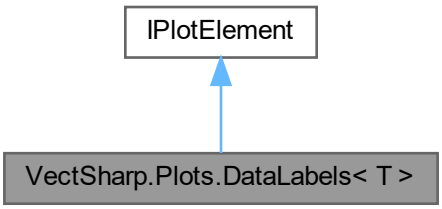
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.33 VectSharp.Plots.DataLabels< T > Class Template Reference

A plot element that draws a text label at each data point.

Inheritance diagram for VectSharp.Plots.DataLabels< T >:



Public Member Functions

- DataLabels (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Creates a new DataLabels<T> instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points at which the labels will be drawn.
- Func< int, T, object > Label `[get, set]`

  A function used to determine the text of the labels to draw. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return either an IEnumerable<T> of FormattedText objects, which will be used as-is, or any other kind of object, which will be converted to a `string` to be used in the label.
- Func< int, T, double > Rotation = (i, d) => 0 `[get, set]`

  A function used to determine the orientation of the labels with respect to the horizontal. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return a `double` representing the angle with respect to the horizontal at which the label should be drawn.
- Func< int, T, Point > Margin = (i, d) => new Point() `[get, set]`

  A function used to determine the position of the labels with respect to the data points. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return a Point defining the amount of space between the data point and the label, in plot coordinates.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the labels.

- TextAnchors Alignment = TextAnchors.Center [get, set]

  The alignment for the labels.
- ICoordinateSystem< T > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = null } [get, set]

  Presentation attributes determining the appearance of the labels.
- string Tag [get, set]

  A tag to identify the labels in the plot.

7.33.1 Detailed Description

A plot element that draws a text label at each data point.

Template Parameters

T	The kind of data describing the data points (generally, IReadOnlyList<double>).

Definition at line 336 of file DataPoints.cs.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 DataLabels()

```
VectSharp.Plots.DataLabels< T >.DataLabels (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Creates a new DataLabels<T> instance.

Parameters

data	The data points at which the labels will be drawn.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 445 of file DataPoints.cs.

7.33.3 Member Function Documentation

7.33.3.1 Plot()

```
void VectSharp.Plots.DataLabels< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 465 of file DataPoints.cs.

7.33.4 Property Documentation

7.33.4.1 Alignment

```
TextAnchors VectSharp.Plots.DataLabels< T >.Alignment = TextAnchors.Center  [get], [set]
```

The alignment for the labels.

Definition at line 422 of file DataPoints.cs.

7.33.4.2 Baseline

```
TextBaselines VectSharp.Plots.DataLabels< T >.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the labels.

Definition at line 417 of file DataPoints.cs.

7.33.4.3 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.DataLabels< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 427 of file DataPoints.cs.

7.33.4.4 Data

```
IEnumerable<T> VectSharp.Plots.DataLabels< T >.Data  [get], [set]
```

The data points at which the labels will be drawn.

Definition at line 341 of file DataPoints.cs.

7.33.4.5 Label

```
Func<int, T, object> VectSharp.Plots.DataLabels< T >.Label  [get], [set]
```

A function used to determine the text of the labels to draw. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return either an IEnumerable<T> of FormattedText objects, which will be used as-is, or any other kind of object, which will be converted to a `string` to be used in the label.

Definition at line 352 of file DataPoints.cs.

7.33.4.6 Margin

```
Func<int, T, Point> VectSharp.Plots.DataLabels< T >.Margin = (i, d) => new Point()  [get],
[set]
```

A function used to determine the position of the labels with respect to the data points. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return a Point defining the amount of space between the data point and the label, in plot coordinates.

Definition at line 412 of file DataPoints.cs.

7.33.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.DataLabels< T >.PresentationAttributes = new
PlotElementPresentationAttributes() { Stroke = null }  [get], [set]
```

Presentation attributes determining the appearance of the labels.

Definition at line 433 of file DataPoints.cs.

7.33.4.8 Rotation

```
Func<int, T, double> VectSharp.Plots.DataLabels< T >.Rotation = (i, d) => 0  [get], [set]
```

A function used to determine the orientation of the labels with respect to the horizontal. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return a `double` representing the angle with respect to the horizontal at which the label should be drawn.

Definition at line 405 of file DataPoints.cs.

7.33.4.9 Tag

```
string VectSharp.Plots.DataLabels< T >.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 438 of file DataPoints.cs.

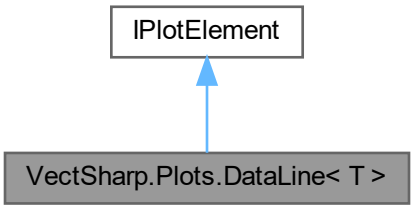
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.34 VectSharp.Plots.DataLine< T > Class Template Reference

A plot element that draws a line passing through a set of points.

Inheritance diagram for VectSharp.Plots.DataLine< T >:



Public Member Functions

- DataLine (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Create a new instance of the DataLine< T > class.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points through which the line will pass.
- bool Smooth `[get, set]`

  If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the line.
- string Tag `[get, set]`

  A tag to identify the labels in the plot.

7.34.1 Detailed Description

A plot element that draws a line passing through a set of points.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 515 of file DataPoints.cs.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 DataLine()

```
VectSharp.Plots.DataLine< T >.DataLine (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new instance of the DataLine<T> class.

Parameters

data	The data points through which the line will pass.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 549 of file DataPoints.cs.

7.34.3 Member Function Documentation

7.34.3.1 Plot()

```
void VectSharp.Plots.DataLine< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 556 of file DataPoints.cs.

7.34.4 Property Documentation

7.34.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.DataLine< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 531 of file DataPoints.cs.

7.34.4.2 Data

```
IEnumerable<T> VectSharp.Plots.DataLine< T >.Data  [get], [set]
```

The data points through which the line will pass.

Definition at line 520 of file DataPoints.cs.

7.34.4.3 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.DataLine< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the line.

Definition at line 537 of file DataPoints.cs.

7.34.4.4 Smooth

```
bool VectSharp.Plots.DataLine< T >.Smooth  [get], [set]
```

If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.

Definition at line 526 of file DataPoints.cs.

7.34.4.5 Tag

```
string VectSharp.Plots.DataLine< T >.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 542 of file DataPoints.cs.

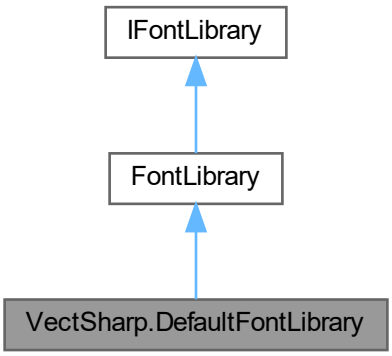
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.35 VectSharp.DefaultFontLibrary Class Reference

A default font library that resolves standard families using the embedded fonts.

Inheritance diagram for VectSharp.DefaultFontLibrary:



Public Member Functions

- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.35.1 Detailed Description

A default font library that resolves standard families using the embedded fonts.

Definition at line 461 of file FontLibrary.cs.

7.35.2 Member Function Documentation

7.35.2.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.DefaultFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 523 of file FontLibrary.cs.

7.35.2.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.DefaultFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 464 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.36 VectSharp.Font.DetailedFontMetrics Class Reference

Represents detailed information about the metrics of a text string when drawn with a certain font.

Properties

- double Width [get]

  Width of the text (measured on the actual glyph outlines).
- double Height [get]

  Height of the text (measured on the actual glyph outlines).
- double LeftSideBearing [get]

  How much the leftmost glyph in the string overhangs the glyph origin on the left. Positive for glyphs that hang past the origin (e.g. italic 'f').
- double RightSideBearing [get]

  How much the rightmost glyph in the string overhangs the glyph end on the right. Positive for glyphs that hang past the end (e.g. italic 'f').
- double Top [get]

  Height of the tallest glyph in the string over the baseline. Always >= 0.
- double Bottom [get]

  Depth of the deepest glyph in the string below the baseline. Always <= 0.
- double AdvanceWidth [get]

  Advance width of the text (excluding any left- or right- side bearing).

7.36.1 Detailed Description

Represents detailed information about the metrics of a text string when drawn with a certain font.

Definition at line 100 of file Font.cs.

7.36.2 Property Documentation

7.36.2.1 AdvanceWidth

```
double VectSharp.Font.DetailedFontMetrics.AdvanceWidth  [get]
```

Advance width of the text (excluding any left- or right- side bearing).

Definition at line 135 of file Font.cs.

7.36.2.2 Bottom

```
double VectSharp.Font.DetailedFontMetrics.Bottom  [get]
```

Depth of the deepest glyph in the string below the baseline. Always <= 0.

Definition at line 130 of file Font.cs.

7.36.2.3 Height

```
double VectSharp.Font.DetailedFontMetrics.Height  [get]
```

Height of the text (measured on the actual glyph outlines).

Definition at line 110 of file Font.cs.

7.36.2.4 LeftSideBearing

```
double VectSharp.Font.DetailedFontMetrics.LeftSideBearing  [get]
```

How much the leftmost glyph in the string overhangs the glyph origin on the left. Positive for glyphs that hang past the origin (e.g. italic 'f').

Definition at line 115 of file Font.cs.

7.36.2.5 RightSideBearing

```
double VectSharp.Font.DetailedFontMetrics.RightSideBearing  [get]
```

How much the rightmost glyph in the string overhangs the glyph end on the right. Positive for glyphs that hang past the end (e.g. italic 'f').

Definition at line 120 of file Font.cs.

7.36.2.6 Top

```
double VectSharp.Font.DetailedFontMetrics.Top  [get]
```

Height of the tallest glyph in the string over the baseline. Always >= 0.

Definition at line 125 of file Font.cs.

7.36.2.7 Width

```
double VectSharp.Font.DetailedFontMetrics.Width  [get]
```

Width of the text (measured on the actual glyph outlines).

Definition at line 105 of file Font.cs.

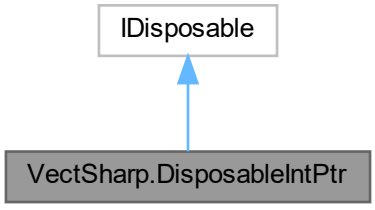
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.37 VectSharp.DisposableIntPtr Class Reference

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Inheritance diagram for VectSharp.DisposableIntPtr:



Public Member Functions

- DisposableIntPtr (IntPtr pointer)

  Create a new DisposableIntPtr.
- void Dispose ()

Public Attributes

- readonly IntPtr InternalPointer

  The pointer to the unmanaged memory.

7.37.1 Detailed Description

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Definition at line 53 of file RasterImage.cs.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 DisposableIntPtr()

```
VectSharp.DisposableIntPtr.DisposableIntPtr (
            IntPtr pointer )
```

Create a new DisposableIntPtr.

Parameters

pointer	The pointer that should be freed upon disposing of this object.

Definition at line 64 of file RasterImage.cs.

7.37.3 Member Function Documentation

7.37.3.1 Dispose()

```
void VectSharp.DisposableIntPtr.Dispose ( )
```

Definition at line 88 of file RasterImage.cs.

7.37.4 Member Data Documentation

7.37.4.1 InternalPointer

```
readonly IntPtr VectSharp.DisposableIntPtr.InternalPointer
```

The pointer to the unmanaged memory.

Definition at line 58 of file RasterImage.cs.

The documentation for this class was generated from the following file:

- VectSharp/RasterImage.cs

7.38 VectSharp.Document Class Reference

Represents a collection of pages.

Public Member Functions

- Document ()
  Create a new document.

Public Attributes

- List< Page > Pages = new List<Page>()
  The pages in the document.

7.38.1 Detailed Description

Represents a collection of pages.

Definition at line 27 of file Document.cs.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 Document()

```
VectSharp.Document.Document ( )
```

Create a new document.

Definition at line 38 of file Document.cs.

7.38.3 Member Data Documentation

7.38.3.1 Pages

```
List<Page> VectSharp.Document.Pages = new List<Page>()
```

The pages in the document.

Definition at line 32 of file Document.cs.

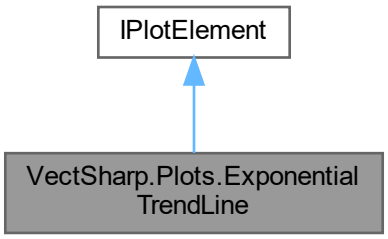
The documentation for this class was generated from the following file:

- VectSharp/Document.cs

7.39 VectSharp.Plots.ExponentialTrendLine Class Reference

A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.

Inheritance diagram for VectSharp.Plots.ExponentialTrendLine:



Public Member Functions

- ExponentialTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new ExponentialTrendLine instance, specifying the equation parameters.
- ExponentialTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.
- ExponentialTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩System, double? fixedIntercept=null)

  Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope `[get, set]`

  The slope of the trendline (a).
- double Intercept `[get, set]`

  The intercept of the trendline (b).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

  The maximum X value for which the trendline is plotted.
- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.39.1 Detailed Description

A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.

Definition at line 256 of file Trendlines.cs.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 ExponentialTrendLine() [1/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ExponentialTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 314 of file Trendlines.cs.

7.39.2.2 ExponentialTrendLine() [2/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 331 of file Trendlines.cs.

7.39.2.3 ExponentialTrendLine() [3/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 385 of file Trendlines.cs.

7.39.3 Member Function Documentation

7.39.3.1 Plot()

```
void VectSharp.Plots.ExponentialTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 388 of file Trendlines.cs.

7.39.4 Property Documentation

7.39.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ExponentialTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 301 of file Trendlines.cs.

7.39.4.2 Intercept

```
double VectSharp.Plots.ExponentialTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 266 of file Trendlines.cs.

7.39.4.3 MaxX

```
double VectSharp.Plots.ExponentialTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 281 of file Trendlines.cs.

7.39.4.4 MaxY

```
double VectSharp.Plots.ExponentialTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 286 of file Trendlines.cs.

7.39.4.5 MinX

```
double VectSharp.Plots.ExponentialTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 271 of file Trendlines.cs.

7.39.4.6 MinY

```
double VectSharp.Plots.ExponentialTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 276 of file Trendlines.cs.

7.39.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ExponentialTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 291 of file Trendlines.cs.

7.39.4.8 Slope

```
double VectSharp.Plots.ExponentialTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 261 of file Trendlines.cs.

7.39.4.9 Tag

```
string VectSharp.Plots.ExponentialTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 296 of file Trendlines.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.40 VectSharp.Canvas.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations
  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)
  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseAllWithSkia, 1, true)
  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseAllWithSkia [get]

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 [get]

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true [get]

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.40.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 1480 of file SKRenderContext.cs.

7.40.2 Member Enumeration Documentation

7.40.2.1 FilterOperations

```
enum VectSharp.Canvas.FilterOption.FilterOperations
```

Defines whether image filters should be rasterised or not.

Definition at line 1485 of file SKRenderContext.cs.

7.40.3 Constructor & Destructor Documentation

7.40.3.1 FilterOption()

```
VectSharp.Canvas.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1534 of file SKRenderContext.cs.

7.40.4 Member Data Documentation

7.40.4.1 Default

```
FilterOption VectSharp.Canvas.FilterOption.Default = new FilterOption(FilterOperations.↩
RasteriseAllWithSkia, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 1526 of file SKRenderContext.cs.

7.40.5 Property Documentation

7.40.5.1 Operation

```
FilterOperations VectSharp.Canvas.FilterOption.Operation = FilterOperations.RasteriseAllWith↩
Skia  [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 1511 of file SKRenderContext.cs.

7.40.5.2 RasterisationResolution

```
double VectSharp.Canvas.FilterOption.RasterisationResolution = 1  [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 1516 of file SKRenderContext.cs.

7.40.5.3 RasterisationResolutionRelative

```
bool VectSharp.Canvas.FilterOption.RasterisationResolutionRelative = true  [get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1521 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.41 VectSharp.PDF.PDFContextInterpreter.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations

  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)

  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseAll, 1, true)

  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseAll [get]

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 [get]

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true [get]

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.41.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 1142 of file PDFContext.cs.

7.41.2 Member Enumeration Documentation

7.41.2.1 FilterOperations

```
enum VectSharp.PDF.PDFContextInterpreter.FilterOption.FilterOperations
```

Defines whether image filters should be rasterised or not.

Definition at line 1147 of file PDFContext.cs.

7.41.3 Constructor & Destructor Documentation

7.41.3.1 FilterOption()

```
VectSharp.PDF.PDFContextInterpreter.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1191 of file PDFContext.cs.

7.41.4 Member Data Documentation

7.41.4.1 Default

```
FilterOption VectSharp.PDF.PDFContextInterpreter.FilterOption.Default = new FilterOption(Filter↩
Operations.RasteriseAll, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 1183 of file PDFContext.cs.

7.41.5 Property Documentation

7.41.5.1 Operation

```
FilterOperations VectSharp.PDF.PDFContextInterpreter.FilterOption.Operation = FilterOperations.↩
RasteriseAll  [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 1168 of file PDFContext.cs.

7.41.5.2 RasterisationResolution

```
double VectSharp.PDF.PDFContextInterpreter.FilterOption.RasterisationResolution = 1  [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 1173 of file PDFContext.cs.

7.41.5.3 RasterisationResolutionRelative

```
bool VectSharp.PDF.PDFContextInterpreter.FilterOption.RasterisationResolutionRelative = true
[get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1178 of file PDFContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFContext.cs

7.42 VectSharp.SVG.SVGContextInterpreter.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations

  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)

  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseIfNecessary, 1, true)

  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseIfNecessary `[get]`

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 `[get]`

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true `[get]`

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.42.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 2280 of file SVGContext.cs.

7.42.2 Member Enumeration Documentation

7.42.2.1 FilterOperations

`enum VectSharp.SVG.SVGContextInterpreter.FilterOption.FilterOperations`

Defines whether image filters should be rasterised or not.

Definition at line 2285 of file SVGContext.cs.

7.42.3 Constructor & Destructor Documentation

7.42.3.1 FilterOption()

```
VectSharp.SVG.SVGContextInterpreter.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 2344 of file SVGContext.cs.

7.42.4 Member Data Documentation

7.42.4.1 Default

```
FilterOption VectSharp.SVG.SVGContextInterpreter.FilterOption.Default = new FilterOption(Filter↩
Operations.RasteriseIfNecessary, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 2336 of file SVGContext.cs.

7.42.5 Property Documentation

7.42.5.1 Operation

```
FilterOperations VectSharp.SVG.SVGContextInterpreter.FilterOption.Operation = FilterOperations.↩
RasteriseIfNecessary  [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 2321 of file SVGContext.cs.

7.42.5.2 RasterisationResolution

```
double VectSharp.SVG.SVGContextInterpreter.FilterOption.RasterisationResolution = 1  [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 2326 of file SVGContext.cs.

7.42.5.3 RasterisationResolutionRelative

```
bool VectSharp.SVG.SVGContextInterpreter.FilterOption.RasterisationResolutionRelative = true
[get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 2331 of file SVGContext.cs.

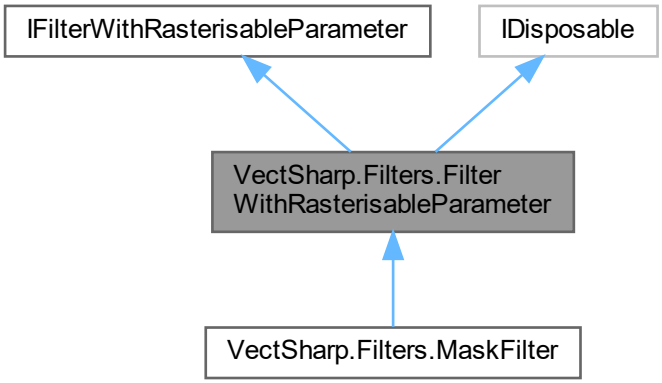
The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGContext.cs

7.43 VectSharp.Filters.FilterWithRasterisableParameter Class Reference

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.

Inheritance diagram for VectSharp.Filters.FilterWithRasterisableParameter:



Public Member Functions

- virtual void RasteriseParameter (Func< Graphics, Rectangle, double, bool, RasterImage > rasterisation↩ Method, double scale)

  Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	*The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.*
scale	*The scale factor at which the parameter is rasterised.*

- void Dispose ()

7.43.1 Detailed Description

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.

Definition at line 86 of file Filters.cs.

7.43.2 Member Function Documentation

7.43.2.1 Dispose()

```
void VectSharp.Filters.FilterWithRasterisableParameter.Dispose ( )
```

Definition at line 187 of file Filters.cs.

7.43.2.2 RasteriseParameter()

```
virtual void VectSharp.Filters.FilterWithRasterisableParameter.RasteriseParameter (
            Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod,
            double scale )  [virtual]
```

Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.
scale	The scale factor at which the parameter is rasterised.

Implements VectSharp.Filters.IFilterWithRasterisableParameter.

Definition at line 135 of file Filters.cs.

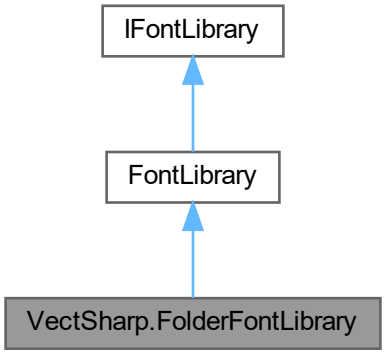
The documentation for this class was generated from the following file:

- VectSharp/Filters/Filters.cs

7.44 VectSharp.FolderFontLibrary Class Reference

A font library that resolves fonts from a folder containing TrueType files.

Inheritance diagram for VectSharp.FolderFontLibrary:



Public Member Functions

- FolderFontLibrary (string folderPath)

  Creates a new FolderFontLibrary using fonts from the specified path, and using the FontFamily.DefaultFontLibrary to resolve the standard font families.
- FolderFontLibrary (string folderPath, IFontLibrary standardFontLibrary)

  Creates a new FolderFontLibrary using fonts from the specified path, and using the specified IFontLibrary to resolve the standard font families.
- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.44.1 Detailed Description

A font library that resolves fonts from a folder containing TrueType files.

Definition at line 555 of file FontLibrary.cs.

7.44.2 Constructor & Destructor Documentation

7.44.2.1 FolderFontLibrary() [1/2]

```
VectSharp.FolderFontLibrary.FolderFontLibrary (
            string folderPath )
```

Creates a new FolderFontLibrary using fonts from the specified path, and using the FontFamily.DefaultFontLibrary to resolve the standard font families.

Parameters

folderPath	The path to the folder containing the TrueType files.

Definition at line 565 of file FontLibrary.cs.

7.44.2.2 FolderFontLibrary() [2/2]

```
VectSharp.FolderFontLibrary.FolderFontLibrary (
            string folderPath,
            IFontLibrary standardFontLibrary )
```

Creates a new FolderFontLibrary using fonts from the specified path, and using the specified IFontLibrary to resolve the standard font families.

Parameters

folderPath	The path to the folder containing the TrueType files.
standardFontLibrary	The IFontLibrary to use when resolving standard font families.

Definition at line 572 of file FontLibrary.cs.

7.44.3 Member Function Documentation

7.44.3.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.FolderFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 636 of file FontLibrary.cs.

7.44.3.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.FolderFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 604 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.45 VectSharp.Font Class Reference

Represents a typeface with a specific size.

Classes

- class DetailedFontMetrics

  Represents detailed information about the metrics of a text string when drawn with a certain font.
- class FontUnderline

  Represents options to underline text.

Public Member Functions

- Font (FontFamily fontFamily, double fontSize)

  Create a new Font object, given the base typeface and the font size.
- Font (FontFamily fontFamily, double fontSize, bool underlined)

  Create a new Font object, given the base typeface, the font size, and a boolean value determining whether text using this font should be underlined.
- Font (FontFamily fontFamily, double fontSize, FontUnderline underline)

  Create a new Font object, given the base typeface, the font size, and an object describing the underline properties of text drawn using this font.
- Size MeasureText (string text)

  Measure the size of a text string when typeset with this font.
- DetailedFontMetrics MeasureTextAdvanced (string text)

  Measure all the metrics of a text string when typeset with this font.

Static Public Attributes

- static bool EnableKerning = true

  Determines whether text kerning is enabled. Note that, even when this is set to `false`, text kerning will be applied on some platforms. For the best consistency, leave this set to `true`.

Properties

- double FontSize `[get]`

  Font size, in graphics units.
- FontFamily FontFamily `[get]`

  Font typeface.
- double Ascent `[get]`

  Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than this). Always >= 0.
- double WinAscent `[get]`

  Height above the baseline for a clipping region (Windows ascent). Always >= 0.
- double Descent `[get]`

  Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than this). Always <= 0.
- double YMax `[get]`

  Absolute maximum height over the baseline of the glyphs in the font. Always >= 0.
- double YMin `[get]`

  Absolute maximum depth below the baseline of the glyphs in the font. Always <= 0.
- FontUnderline Underline `[get]`

  Determines the underline style of text drawn using this font. If this is `null`, the text is not underlined.

7.45.1 Detailed Description

Represents a typeface with a specific size.

Definition at line 27 of file Font.cs.

7.45.2 Constructor & Destructor Documentation

7.45.2.1 Font() [1/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize )
```

Create a new Font object, given the base typeface and the font size.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.

Definition at line 164 of file Font.cs.

7.45.2.2 Font() [2/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize,
            bool underlined )
```

Create a new Font object, given the base typeface, the font size, and a boolean value determining whether text using this font should be underlined.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.
underlined	A boolean value determining whether text drawn using this font should be underlined. The appearance of the underline can be tweaked by changing the properties of the Underline property after the font has been created.

Definition at line 176 of file Font.cs.

7.45.2.3 Font() [3/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize,
            FontUnderline underline )
```

Create a new Font object, given the base typeface, the font size, and an object describing the underline properties of text drawn using this font.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.
underline	A FontUnderline object describing the underline properties of text drawn using this font.

Definition at line 193 of file Font.cs.

7.45.3 Member Function Documentation

7.45.3.1 MeasureText()

```
Size VectSharp.Font.MeasureText (
            string text )
```

Measure the size of a text string when typeset with this font.

Parameters

text	The string to measure.

Returns

A Size object representing the width and height of the text.

Definition at line 300 of file Font.cs.

7.45.3.2 MeasureTextAdvanced()

```
DetailedFontMetrics VectSharp.Font.MeasureTextAdvanced (
            string text )
```

Measure all the metrics of a text string when typeset with this font.

Parameters

text	The string to measure.

Returns

A DetailedFontMetrics object representing the metrics of the text.

Definition at line 357 of file Font.cs.

7.45.4 Member Data Documentation

7.45.4.1 EnableKerning

```
bool VectSharp.Font.EnableKerning = true  [static]
```

Determines whether text kerning is enabled. Note that, even when this is set to `false`, text kerning will be applied on some platforms. For the best consistency, leave this set to `true`.

Definition at line 32 of file Font.cs.

7.45.5 Property Documentation

7.45.5.1 Ascent

```
double VectSharp.Font.Ascent  [get]
```

Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than this). Always >= 0.

Definition at line 203 of file Font.cs.

7.45.5.2 Descent

```
double VectSharp.Font.Descent  [get]
```

Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than this). Always <= 0.

Definition at line 239 of file Font.cs.

7.45.5.3 FontFamily

```
FontFamily VectSharp.Font.FontFamily  [get]
```

Font typeface.

Definition at line 157 of file Font.cs.

7.45.5.4 FontSize

```
double VectSharp.Font.FontSize [get]
```

Font size, in graphics units.

Definition at line 152 of file Font.cs.

7.45.5.5 Underline

```
FontUnderline VectSharp.Font.Underline [get]
```

Determines the underline style of text drawn using this font. If this is `null`, the text is not underlined.

Definition at line 293 of file Font.cs.

7.45.5.6 WinAscent

```
double VectSharp.Font.WinAscent [get]
```

Height above the baseline for a clipping region (Windows ascent). Always >= 0.

Definition at line 221 of file Font.cs.

7.45.5.7 YMax

```
double VectSharp.Font.YMax [get]
```

Absolute maximum height over the baseline of the glyphs in the font. Always >= 0.

Definition at line 257 of file Font.cs.

7.45.5.8 YMin

```
double VectSharp.Font.YMin [get]
```

Absolute maximum depth below the baseline of the glyphs in the font. Always <= 0.

Definition at line 275 of file Font.cs.

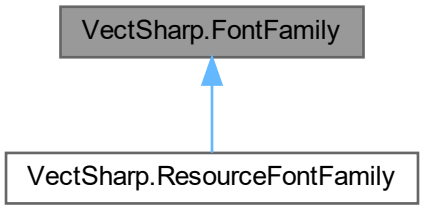
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.46 VectSharp.FontFamily Class Reference

Represents a typeface.

Inheritance diagram for VectSharp.FontFamily:



Public Types

- enum StandardFontFamilies

  The 14 standard font families.

Public Member Functions

- FontFamily (string fileName)

  Create a new FontFamily.
- FontFamily (Stream ttfStream)

  Create a new FontFamily.
- FontFamily (TrueTypeFile ttf)

  Create a new FontFamily.
- FontFamily (StandardFontFamilies standardFontFamily)

  Create a new standard FontFamily.

Static Public Member Functions

- static FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (string fontFamily, StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback . Equivalent to DefaultFontLibrary.ResolveFontFamily.

Static Public Attributes

- static string[ ] StandardFamilies = new string[ ] { "Times-Roman", "Times-Bold", "Times-Italic", "Times-Bold↩ Italic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique", "Helvetica-BoldOblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique", "Symbol", "ZapfDingbats" }

  The names of the 14 standard families that are guaranteed to be displayed correctly.
- static string[ ] StandardFontFamilyResources

  The names of the resource streams pointing to the included TrueType font files for each of the standard 14 font families.

Properties

- static IFontLibrary DefaultFontLibrary = new DefaultFontLibrary() `[get, set]`

  The default font library used to resolve font family names.
- bool IsStandardFamily `[get]`

  Whether this is one of the 14 standard font families or not.
- string FileName `[get]`

  Full path to the TrueType font file for this font family (or, if this is a standard font family, name of the font family).
- string FamilyName `[get]`

  Name of the font family, including any variantes.
- TrueTypeFile TrueTypeFile `[get]`

  Parsed TrueType font file for this font family. See also:
  See also

  VectSharp.TrueTypeFile

  .
- bool IsBold `[get]`

  Whether this font is bold or not. This is set based on the information included in the OS/2 table of the TrueType file.
- bool IsItalic `[get]`

  Whether this font is italic or oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.
- bool IsOblique `[get]`

  Whether this font is oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

7.46.1 Detailed Description

Represents a typeface.

Definition at line 420 of file Font.cs.

7.46.2 Member Enumeration Documentation

7.46.2.1 StandardFontFamilies

`enum VectSharp.FontFamily.StandardFontFamilies`

The 14 standard font families.

Definition at line 499 of file Font.cs.

7.46.3 Constructor & Destructor Documentation

7.46.3.1 FontFamily() [1/4]

```
VectSharp.FontFamily.FontFamily (
            string fileName )
```

Create a new FontFamily.

Parameters

fileName	The full path to the TrueType font file for this font family or the name of a standard font family.

Definition at line 608 of file Font.cs.

7.46.3.2 FontFamily() [2/4]

```
VectSharp.FontFamily.FontFamily (
            Stream ttfStream )
```

Create a new FontFamily.

Parameters

ttfStream	A stream containing a file in TTF format.

Definition at line 633 of file Font.cs.

7.46.3.3 FontFamily() [3/4]

```
VectSharp.FontFamily.FontFamily (
            TrueTypeFile ttf )
```

Create a new FontFamily.

Parameters

ttf	A font file in TTF format.

Definition at line 653 of file Font.cs.

7.46.3.4 FontFamily() [4/4]

```
VectSharp.FontFamily.FontFamily (
            StandardFontFamilies standardFontFamily )
```

Create a new standard FontFamily.

Parameters

standardFontFamily	The standard font family.

Definition at line 674 of file Font.cs.

7.46.4 Member Function Documentation

7.46.4.1 ResolveFontFamily() [1/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            StandardFontFamilies standardFontFamily )  [static]
```

Create a new font family from the specified standard font family name. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

7.46.4.2 ResolveFontFamily() [2/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

7.46.4.3 ResolveFontFamily() [3/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.46.4.4 ResolveFontFamily() [4/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily,
            StandardFontFamilies finalFallback,
            params string[] fallback )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* . Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.46.5 Member Data Documentation

7.46.5.1 StandardFamilies

```
string [] VectSharp.FontFamily.StandardFamilies = new string[] { "Times-Roman", "Times-Bold",
"Times-Italic", "Times-BoldItalic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique", "Helvetica-Bold↩
Oblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique", "Symbol", "Zapf↩
Dingbats" }  [static]
```

The names of the 14 standard families that are guaranteed to be displayed correctly.

Definition at line 478 of file Font.cs.

7.46.5.2 StandardFontFamilyResources

```
string [] VectSharp.FontFamily.StandardFontFamilyResources  [static]
```

Initial value:

```
= new string[]
        {
            "VectSharp.StandardFonts.Tinos-Regular.ttf", "VectSharp.StandardFonts.Tinos-Bold.ttf",
      "VectSharp.StandardFonts.Tinos-Italic.ttf", "VectSharp.StandardFonts.Tinos-BoldItalic.ttf",
            "VectSharp.StandardFonts.Arimo-Regular.ttf", "VectSharp.StandardFonts.Arimo-Bold.ttf",
      "VectSharp.StandardFonts.Arimo-Italic.ttf", "VectSharp.StandardFonts.Arimo-BoldItalic.ttf",
            "VectSharp.StandardFonts.Cousine-Regular.ttf", "VectSharp.StandardFonts.Cousine-Bold.ttf",
      "VectSharp.StandardFonts.Cousine-Italic.ttf", "VectSharp.StandardFonts.Cousine-BoldItalic.ttf",
            "VectSharp.StandardFonts.SymbolNeu_GB.ttf", "VectSharp.StandardFonts.Levibats-Regular_GB.ttf"
        }
```

The names of the resource streams pointing to the included TrueType font files for each of the standard 14 font families.

Definition at line 483 of file Font.cs.

7.46.6 Property Documentation

7.46.6.1 DefaultFontLibrary

```
IFontLibrary VectSharp.FontFamily.DefaultFontLibrary = new DefaultFontLibrary()  [static],
[get], [set]
```

The default font library used to resolve font family names.

Definition at line 425 of file Font.cs.

7.46.6.2 FamilyName

```
string VectSharp.FontFamily.FamilyName  [get]
```

Name of the font family, including any variantes.

Definition at line 580 of file Font.cs.

7.46.6.3 FileName

```
string VectSharp.FontFamily.FileName  [get]
```

Full path to the TrueType font file for this font family (or, if this is a standard font family, name of the font family).

Definition at line 575 of file Font.cs.

7.46.6.4 IsBold

```
bool VectSharp.FontFamily.IsBold  [get]
```

Whether this font is bold or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 591 of file Font.cs.

7.46.6.5 IsItalic

```
bool VectSharp.FontFamily.IsItalic  [get]
```

Whether this font is italic or oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 596 of file Font.cs.

7.46.6.6 IsOblique

```
bool VectSharp.FontFamily.IsOblique  [get]
```

Whether this font is oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 601 of file Font.cs.

7.46.6.7 IsStandardFamily

```
bool VectSharp.FontFamily.IsStandardFamily  [get]
```

Whether this is one of the 14 standard font families or not.

Definition at line 494 of file Font.cs.

7.46.6.8 TrueTypeFile

```
TrueTypeFile VectSharp.FontFamily.TrueTypeFile  [get]
```

Parsed TrueType font file for this font family. See also:

See also

VectSharp.TrueTypeFile

.

Definition at line 586 of file Font.cs.

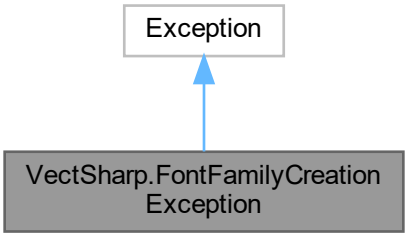
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.47 VectSharp.FontFamilyCreationException Class Reference

An exception that occurs while creating a FontFamily.

Inheritance diagram for VectSharp.FontFamilyCreationException:



Public Member Functions

- FontFamilyCreationException (string fontFamily)

  Create a new FontFamilyCreationException instance.

Properties

- string FontFamily `[get]`

  The name of the font family that was being created.

7.47.1 Detailed Description

An exception that occurs while creating a FontFamily.

Definition at line 441 of file FontLibrary.cs.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 FontFamilyCreationException()

```
VectSharp.FontFamilyCreationException.FontFamilyCreationException (
            string fontFamily )
```

Create a new FontFamilyCreationException instance.

Parameters

fontFamily	The name of the font family that was being created.

Definition at line 452 of file FontLibrary.cs.

7.47.3 Property Documentation

7.47.3.1 FontFamily

```
string VectSharp.FontFamilyCreationException.FontFamily [get]
```

The name of the font family that was being created.

Definition at line 446 of file FontLibrary.cs.

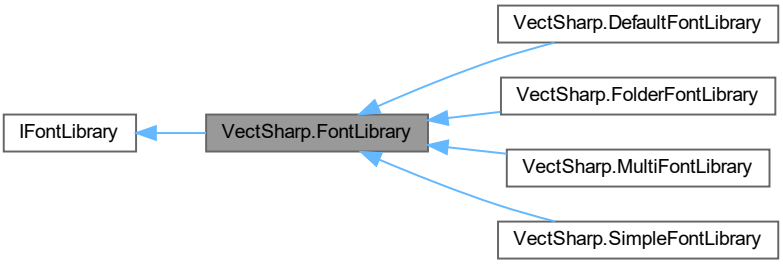
The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.48 VectSharp.FontLibrary Class Reference

Abstract class with a default implementation of font family fallbacks.

Inheritance diagram for VectSharp.FontLibrary:



Public Member Functions

- abstract FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- abstract FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

- virtual FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*
fallback	*Names of additional font families or TTF files, which will be tried if the first fontFamily is not valid.*

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

- virtual FontFamily ResolveFontFamily (string fontFamily, FontFamily.StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback .

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*
fallback	*Names of additional font families or TTF files, which will be tried if the first fontFamily is not valid.*
finalFallback	*The standard name of the font family that will be used if none of the fallback families are valid.*

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.48.1 Detailed Description

Abstract class with a default implementation of font family fallbacks.

Definition at line 67 of file FontLibrary.cs.

7.48.2 Member Function Documentation

7.48.2.1 ResolveFontFamily() [1/4]

```
abstract FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [pure virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.IFontLibrary.

Implemented in VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.48.2.2 ResolveFontFamily() [2/4]

```
abstract FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily )  [pure virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.IFontLibrary.

Implemented in VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.48.2.3 ResolveFontFamily() [3/4]

```
virtual FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily,
            FontFamily.StandardFontFamilies finalFallback,
            params string[] fallback )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* .

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implements VectSharp.IFontLibrary.

Definition at line 129 of file FontLibrary.cs.

7.48.2.4 ResolveFontFamily() [4/4]

```
virtual FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implements VectSharp.IFontLibrary.

Definition at line 76 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.49 VectSharp.Font.FontUnderline Class Reference

Represents options to underline text.

Properties

- bool SkipDescenders [get, set]

  Determines whether the underline skips the parts of the glyph that would intersect with it or not.
- double Position [get, set]

  Determines the position of the top of the underline with respect to the text baseline. Positive values are below the baseline, negative values are above it. This is expressed as a fraction of the font size.
- double Thickness [get, set]

  Determines the thickness of the underline, expressed as a fraction of the font size.
- LineCaps LineCap [get, set]

  Determines the caps at the start and end of the underline.
- bool FollowItalicAngle [get, set]

  Determine whether the shape of the underline is slanted to follow the angle of italic fonts.

7.49.1 Detailed Description

Represents options to underline text.

Definition at line 37 of file Font.cs.

7.49.2 Property Documentation

7.49.2.1 FollowItalicAngle

```
bool VectSharp.Font.FontUnderline.FollowItalicAngle  [get], [set]
```

Determine whether the shape of the underline is slanted to follow the angle of italic fonts.

Definition at line 62 of file Font.cs.

7.49.2.2 LineCap

```
LineCaps VectSharp.Font.FontUnderline.LineCap  [get], [set]
```

Determines the caps at the start and end of the underline.

Definition at line 57 of file Font.cs.

7.49.2.3 Position

```
double VectSharp.Font.FontUnderline.Position  [get], [set]
```

Determines the position of the top of the underline with respect to the text baseline. Positive values are below the baseline, negative values are above it. This is expressed as a fraction of the font size.

Definition at line 47 of file Font.cs.

7.49.2.4 SkipDescenders

```
bool VectSharp.Font.FontUnderline.SkipDescenders  [get], [set]
```

Determines whether the underline skips the parts of the glyph that would intersect with it or not.

Definition at line 42 of file Font.cs.

7.49.2.5 Thickness

```
double VectSharp.Font.FontUnderline.Thickness  [get], [set]
```

Determines the thickness of the underline, expressed as a fraction of the font size.

Definition at line 52 of file Font.cs.

The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.50 VectSharp.Markdown.FormattedString Struct Reference

Represents a string with associated formatting information.

Public Member Functions

- FormattedString (string text, Colour colour, bool isBold, bool isItalic)
  Creates a new FormattedString instance.

Properties

- string Text [get]
  The text represented by this object.
- Colour Colour [get]
  The colour of the text.
- bool IsBold [get]
  Whether the text should be rendered as bold or not.
- bool IsItalic [get]
  Whether the text should be rendered as italic or not.

7.50.1 Detailed Description

Represents a string with associated formatting information.

Definition at line 32 of file SyntaxHighlighting.cs.

7.50.2 Constructor & Destructor Documentation

7.50.2.1 FormattedString()

```
VectSharp.Markdown.FormattedString.FormattedString (
            string text,
            Colour colour,
            bool isBold,
            bool isItalic )
```

Creates a new FormattedString instance.

Parameters

text	The text of the object.
colour	The colour of the text.
isBold	Whether the text should be rendered as bold or not.
isItalic	Whether the text should be rendered as italic or not.

Definition at line 61 of file SyntaxHighlighting.cs.

7.50.3 Property Documentation

7.50.3.1 Colour

```
Colour VectSharp.Markdown.FormattedString.Colour  [get]
```

The colour of the text.

Definition at line 42 of file SyntaxHighlighting.cs.

7.50.3.2 IsBold

```
bool VectSharp.Markdown.FormattedString.IsBold  [get]
```

Whether the text should be rendered as bold or not.

Definition at line 47 of file SyntaxHighlighting.cs.

7.50.3.3 IsItalic

```
bool VectSharp.Markdown.FormattedString.IsItalic  [get]
```

Whether the text should be rendered as italic or not.

Definition at line 52 of file SyntaxHighlighting.cs.

7.50.3.4 Text

```
string VectSharp.Markdown.FormattedString.Text  [get]
```

The text represented by this object.

Definition at line 37 of file SyntaxHighlighting.cs.

The documentation for this struct was generated from the following file:

- VectSharp.Markdown/SyntaxHighlighting.cs

7.51 VectSharp.FormattedText Class Reference

Represents a run of text that should be drawn with the same style.

Public Member Functions

- FormattedText (string text, Font font, Script script=Script.Normal, Brush brush=null)

  Creates a new FormattedText instance with the specified text , font , script position and brush .

Static Public Member Functions

- static IEnumerable< FormattedText > Format (string text, Font normalFont, Font boldFont, Font italicFont, Font boldItalicFont, Brush defaultBrush=null)

  Parse the formatting information contained in a text string into a collection of FormattedText objects.
- static IEnumerable< FormattedText > Format (string text, FontFamily.StandardFontFamilies fontFamily, double fontSize, bool defaultUnderline=false, Brush defaultBrush=null)

  Parse the formatting information contained in a text string into a collection of FormattedText objects, using fonts from a standard font family.

Properties

- string Text [get]

  Represents the text represented by this instance.
- Font Font [get]

  Represents the font that should be used to draw the text.
- Script Script [get]

  Represents the position of the text.
- Brush Brush [get]

  Represents the brush that should be used to draw the text. If this is null, the default brush is used.

7.51.1 Detailed Description

Represents a run of text that should be drawn with the same style.

Definition at line 50 of file FormattedText.cs.

7.51.2 Constructor & Destructor Documentation

7.51.2.1 FormattedText()

```
VectSharp.FormattedText.FormattedText (
            string text,
            Font font,
            Script script = Script.Normal,
            Brush brush = null )
```

Creates a new FormattedText instance with the specified *text* , *font* , *script* position and *brush* .

Parameters

text	The text that will be contained in the new FormattedText.
font	The font that will be used by the new FormattedText.
script	The script position of the new FormattedText.
brush	The brush that will be used by the new FormattedText.

Definition at line 79 of file FormattedText.cs.

7.51.3 Member Function Documentation

7.51.3.1 Format() [1/2]

```
static IEnumerable< FormattedText > VectSharp.FormattedText.Format (
            string text,
            Font normalFont,
            Font boldFont,
            Font italicFont,
            Font boldItalicFont,
            Brush defaultBrush = null )  [static]
```

Parse the formatting information contained in a text string into a collection of FormattedText objects.

Parameters

text	The string containing formatting information. Format information is specified using HTML-like tags: • or are used for bold text; • or are used for text in italics; • <u></u> are used for underlined text; • and are used, respectively, for superscript and subscript text; • <#COLOUR></#> is used to specify the colour of the text, where COLOUR is a CSS colour string (e.g. <#red>, <#0080FF>, or <#rgba(128, 80, 52, 0.5)>).
normalFont	The font that will be used for text that is neither bold nor italic.
boldFont	The font that will be used for text that is bold. Note that this does not necessarily have to be a bold font; this is just the font that is applied to text contained within tags.
italicFont	The font that will be used for text that is in italics. Note that this does not necessarily have to be an italic font; this is just the font that is applied to text contained within tags.
boldItalicFont	The font that will be used for text that is both in bold and in italics.
defaultBrush	The default Brush that will be used for text runs that do not specify a colour. If this is null, the default Brush will be the one specified in the painting call.

Returns

A lazy collection of FormattedText objects. Note that every enumeration of this collection causes the text to be parsed again; if you need to enumerate this collection more than once, you should probably convert it e.g. to a List<T>.

Definition at line 105 of file FormattedText.cs.

7.51.3.2 Format() [2/2]

```
static IEnumerable< FormattedText > VectSharp.FormattedText.Format (
            string text,
            FontFamily.StandardFontFamilies fontFamily,
            double fontSize,
            bool defaultUnderline = false,
            Brush defaultBrush = null )  [static]
```

Parse the formatting information contained in a text string into a collection of FormattedText objects, using fonts from a standard font family.

Parameters

text	The string containing formatting information. Format information is specified using HTML-like tags: • `<b></b>` or `<strong></strong>` are used for bold text; • `<i></i>` or `<em></em>` are used for text in italics; • `<u></u>` are used for underlined text; • `<sup></sup>` and `<sub></sub>` are used, respectively, for superscript and subscript text; • `<#COLOUR></#>` is used to specify the colour of the text, where `COLOUR` is a CSS colour string (e.g. `<#red>`, `<#0080FF>`, or `<#rgba(128, 80, 52, 0.5)>`).
fontFamily	The font family from which the fonts will be created. If this is a regular font family, the bold, italic and bold-italic versions of the font will be used for the formatted text. Otherwise, the relevant font styles will be toggled (e.g. if the supplied font family is bold, then regular text in the formatted string will be displayed as bold, while bold text in the formatted string will be displayed as regular text).
fontSize	The size of the fonts to use.
defaultUnderline	Determines whether text should be underlined by default. This is toggled by `<u></u>` tags.
defaultBrush	The default Brush that will be used for text runs that do not specify a colour. If this is `null`, the default Brush will be the one specified in the painting call.

Returns

A lazy collection of FormattedText objects. Note that every enumeration of this collection causes the text to be parsed again; if you need to enumerate this collection more than once, you should probably convert it e.g. to a List<T>.

Definition at line 287 of file FormattedText.cs.

7.51.4 Property Documentation

7.51.4.1 Brush

```
Brush VectSharp.FormattedText.Brush  [get]
```

Represents the brush that should be used to draw the text. If this is null, the default brush is used.

Definition at line 70 of file FormattedText.cs.

7.51.4.2 Font

```
Font VectSharp.FormattedText.Font  [get]
```

Represents the font that should be used to draw the text.

Definition at line 60 of file FormattedText.cs.

7.51.4.3 Script

```
Script VectSharp.FormattedText.Script  [get]
```

Represents the position of the text.

Definition at line 65 of file FormattedText.cs.

7.51.4.4 Text

```
string VectSharp.FormattedText.Text  [get]
```

Represents the text represented by this instance.

Definition at line 55 of file FormattedText.cs.

The documentation for this class was generated from the following file:

- VectSharp/FormattedText.cs

7.52 VectSharp.FormattedTextExtensions Class Reference

Contains extension methods for collections of FormattedText objects.

Static Public Member Functions

- static Font.DetailedFontMetrics Measure (this IEnumerable< FormattedText > text)
  Measures a collection of FormattedText objects.
- static string GetText (this IEnumerable< FormattedText > text)
  Extracts the text from a collection of FormattedText objects.

7.52.1 Detailed Description

Contains extension methods for collections of FormattedText objects.

Definition at line 542 of file FormattedText.cs.

7.52.2 Member Function Documentation

7.52.2.1 GetText()

```
static string VectSharp.FormattedTextExtensions.GetText (
            this IEnumerable< FormattedText > text )  [static]
```

Extracts the text from a collection of FormattedText objects.

Parameters

text	The collection of FormattedText objects whose text should be extracted.

Returns

A text rappresentation of the collection of FormattedText objects.

Definition at line 644 of file FormattedText.cs.

7.52.2.2 Measure()

```
static Font.DetailedFontMetrics VectSharp.FormattedTextExtensions.Measure (
            this IEnumerable< FormattedText > text )  [static]
```

Measures a collection of FormattedText objects.

Parameters

text	The collection of FormattedText objects to be measured.

Returns

A Font.DetailedFontMetrics containing detailed measurements for the text obtained by composing the elements in the FormattedText collection.

Definition at line 634 of file FormattedText.cs.

The documentation for this class was generated from the following file:

- VectSharp/FormattedText.cs

7.53 VectSharp.Frame Class Reference

A key frame for an animation.

Public Member Functions

- Frame (Graphics graphics, double duration)

  Creates a new Frame with the specified contents and duration.

Properties

- double Duration [get]

  The duration of the frame, in milliseconds.
- Graphics Graphics [get]

  The contents of the frame.

7.53.1 Detailed Description

A key frame for an animation.

Definition at line 1193 of file Animation.cs.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 Frame()

```
VectSharp.Frame.Frame (
            Graphics graphics,
            double duration )
```

Creates a new Frame with the specified contents and duration.

Parameters

graphics	The contents of the frame.
duration	The duration of the frame, in milliseconds.

Definition at line 1210 of file Animation.cs.

7.53.3 Property Documentation

7.53.3.1 Duration

```
double VectSharp.Frame.Duration  [get]
```

The duration of the frame, in milliseconds.

Definition at line 1198 of file Animation.cs.

7.53.3.2 Graphics

```
Graphics VectSharp.Frame.Graphics  [get]
```

The contents of the frame.

Definition at line 1203 of file Animation.cs.

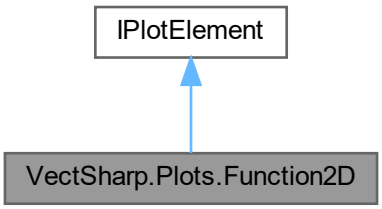
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.54 VectSharp.Plots.Function2D Class Reference

A plot element that plots a function of two variables.

Inheritance diagram for VectSharp.Plots.Function2D:



Public Types

- enum PlotType

  Describes the kind of plots that can be produced.

Public Member Functions

- Function2D (Function2DGrid function, IContinuousInvertibleCoordinateSystem coordinateSystem)

  Create a new Function2D instance.
- unsafe void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IDataPointElement SampledPointElement = new PathDataPointElement() `[get, set]`

  The symbol to draw at the sampled points.
- int RasterResolutionX = 512 `[get, set]`

  Resolution on the X axis for the rasterised tesselation.
- int RasterResolutionY = 512 `[get, set]`

  Resolution on the Y axis for the rasterised tesselation.
- Function2DGrid Function `[get, set]`

  The function to plot.
- PlotType Type = PlotType.SampledPoints `[get, set]`

  The kind of plot that is produced.

- Func< double, Colour > Colouring = x => { x = Math.Max(0, Math.Min(1, x)); return Colour.FromRgb(x, x, x); } `[get, set]`

  A function associating sampled function values to a Colour. You should set this to a function accepting a single `double` argument ranging between 0 and 1, and returning the corresponding colour. The default value returns black 0 and white for 1.
- string Tag `[get, set]`

  A tag to identify the function in the plot.
- IContinuousInvertibleCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.54.1 Detailed Description

A plot element that plots a function of two variables.

Definition at line 520 of file Function2D.cs.

7.54.2 Member Enumeration Documentation

7.54.2.1 PlotType

```
enum VectSharp.Plots.Function2D.PlotType
```

Describes the kind of plots that can be produced.

Definition at line 525 of file Function2D.cs.

7.54.3 Constructor & Destructor Documentation

7.54.3.1 Function2D()

```
VectSharp.Plots.Function2D.Function2D (
            Function2DGrid function,
            IContinuousInvertibleCoordinateSystem coordinateSystem )
```

Create a new Function2D instance.

Parameters

function	The function to plot.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 594 of file Function2D.cs.

7.54.4 Member Function Documentation

7.54.4.1 Plot()

```
unsafe void VectSharp.Plots.Function2D.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 601 of file Function2D.cs.

7.54.5 Property Documentation

7.54.5.1 Colouring

```
Func<double, Colour> VectSharp.Plots.Function2D.Colouring = x => { x = Math.Max(0, Math.↩
Min(1, x)); return Colour.FromRgb(x, x, x); }  [get], [set]
```

A function associating sampled function values to a Colour. You should set this to a function accepting a single `double` argument ranging between 0 and 1, and returning the corresponding colour. The default value returns black 0 and white for 1.

Definition at line 576 of file Function2D.cs.

7.54.5.2 CoordinateSystem

```
IContinuousInvertibleCoordinateSystem VectSharp.Plots.Function2D.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 586 of file Function2D.cs.

7.54.5.3 Function

```
Function2DGrid VectSharp.Plots.Function2D.Function  [get], [set]
```

The function to plot.

Definition at line 564 of file Function2D.cs.

7.54.5.4 RasterResolutionX

```
int VectSharp.Plots.Function2D.RasterResolutionX = 512  [get], [set]
```

Resolution on the X axis for the rasterised tesselation.

Definition at line 554 of file Function2D.cs.

7.54.5.5 RasterResolutionY

```
int VectSharp.Plots.Function2D.RasterResolutionY = 512  [get], [set]
```

Resolution on the Y axis for the rasterised tesselation.

Definition at line 559 of file Function2D.cs.

7.54.5.6 SampledPointElement

```
IDataPointElement VectSharp.Plots.Function2D.SampledPointElement = new PathDataPointElement()
[get], [set]
```

The symbol to draw at the sampled points.

Definition at line 549 of file Function2D.cs.

7.54.5.7 Tag

```
string VectSharp.Plots.Function2D.Tag  [get], [set]
```

A tag to identify the function in the plot.

Definition at line 581 of file Function2D.cs.

7.54.5.8 Type

```
PlotType VectSharp.Plots.Function2D.Type = PlotType.SampledPoints  [get], [set]
```

The kind of plot that is produced.

Definition at line 569 of file Function2D.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Function2D.cs

7.55 VectSharp.Plots.Function2DGrid Class Reference

Represents a function of two variables that has been sampled in some points.

Public Types

- enum GridType

  Describes the arrangement of the points that have been sampled.

Public Member Functions

- Function2DGrid (IReadOnlyList< IReadOnlyList< double > > dataPoints)

  Create a new Function2DGrid from a list of sampled values.
- Function2DGrid (Func< double[ ], double > function, double minX, double minY, double maxX, double maxY, int stepsX, int stepsY, GridType type)

  Create a new Function2DGrid by sampling the provided function.
- Function2DGrid ToRectangular ()

  Converts a hexagonal grid into a rectangular grid.

Properties

- IReadOnlyList< IReadOnlyList< double > > DataPoints [get]

  The points where the function has been sampled.
- double MinX [get]

  The minimum X value that has been sampled.
- double MaxX [get]

  The maximum X value that has been sampled.
- double MinY [get]

  The minimum Y value that has been sampled.
- double MaxY [get]

  The maximum Y value that has been sampled.
- double MinZ [get]

  The minimum value that has been obtained for the function.
- double MaxZ [get]

  The maximum value that has been obtained for the function.
- int StepsX [get]

  If the function has been sampled along a regular grid, the number of steps between MinX and MaxX on the X axis.
- int StepsY [get]

  If the function has been sampled along a regular grid, the number of steps between MinY and MaxY on the X axis.
- GridType Type [get]

  The type of grid.

7.55.1 Detailed Description

Represents a function of two variables that has been sampled in some points.

Definition at line 29 of file Function2D.cs.

7.55.2 Member Enumeration Documentation

7.55.2.1 GridType

```
enum VectSharp.Plots.Function2DGrid.GridType
```

Describes the arrangement of the points that have been sampled.

Definition at line 34 of file Function2D.cs.

7.55.3 Constructor & Destructor Documentation

7.55.3.1 Function2DGrid() [1/2]

```
VectSharp.Plots.Function2DGrid.Function2DGrid (
            IReadOnlyList< IReadOnlyList< double > > dataPoints )
```

Create a new Function2DGrid from a list of sampled values.

Parameters

dataPoints	The sampled values. Each element of this IReadOnlyList<T> should be a collection of three values: the X coordinate of the sampled point, the Y coordinate of the sampled point, and the value of the function at that point.

Definition at line 117 of file Function2D.cs.

7.55.3.2 Function2DGrid() [2/2]

```
VectSharp.Plots.Function2DGrid.Function2DGrid (
            Func< double[], double > function,
            double minX,
            double minY,
            double maxX,
```

```
double maxY,
int stepsX,
int stepsY,
GridType type )
```

Create a new Function2DGrid by sampling the provided function.

Parameters

function	The function to sample.
minX	The minimum X value at which the function should be sampled.
minY	The minimum Y value at which the function should be sampled.
maxX	The maximum X value at which the function should be sampled.
maxY	The maximum Y value at which the function should be sampled.
stepsX	If *type* is not GridType.Irregular, the number of steps between *minX* and *maxX* on the X axis. Otherwise, the number of sampled points is determined by multiplying *stepsX* and *stepsY* together.
stepsY	If *type* is not GridType.Irregular, the number of steps between *minY* and *maxY* on the Y axis. Otherwise, the number of sampled points is determined by multiplying *stepsX* and *stepsY* together.
type	The strategy used to select points to sample. If this is GridType.Irregular, uniformly distributed random points between (*minX* , *minY*) and (*maxX* , *maxY*) are sampled.

Definition at line 163 of file Function2D.cs.

7.55.4 Member Function Documentation

7.55.4.1 ToRectangular()

```
Function2DGrid VectSharp.Plots.Function2DGrid.ToRectangular ( )
```

Converts a hexagonal grid into a rectangular grid.

Returns

If Type is GridType.Rectangular, this method returns the current instance.
If Type is GridType.HexagonHorizontal or GridType.HexagonVertical, a new Function2DGrid with Type equal to GridType.Rectangular is returned. The sampled points in this grid are obtained by performing a bilinear interpolation on the sampled points from this grid. The returned grid will always be "denser" than the current instance.
If Type is GridType.Irregular, an InvalidOperationException is thrown.

Exceptions

InvalidOperationException	Thrown if Type is GridType.Irregular.

Definition at line 322 of file Function2D.cs.

7.55.5 Property Documentation

7.55.5.1 DataPoints

```
IReadOnlyList<IReadOnlyList<double> > VectSharp.Plots.Function2DGrid.DataPoints [get]
```

The points where the function has been sampled.

Definition at line 62 of file Function2D.cs.

7.55.5.2 MaxX

```
double VectSharp.Plots.Function2DGrid.MaxX [get]
```

The maximum X value that has been sampled.

Definition at line 72 of file Function2D.cs.

7.55.5.3 MaxY

```
double VectSharp.Plots.Function2DGrid.MaxY [get]
```

The maximum Y value that has been sampled.

Definition at line 82 of file Function2D.cs.

7.55.5.4 MaxZ

```
double VectSharp.Plots.Function2DGrid.MaxZ [get]
```

The maximum value that has been obtained for the function.

Definition at line 92 of file Function2D.cs.

7.55.5.5 MinX

```
double VectSharp.Plots.Function2DGrid.MinX [get]
```

The minimum X value that has been sampled.

Definition at line 67 of file Function2D.cs.

7.55.5.6 MinY

```
double VectSharp.Plots.Function2DGrid.MinY  [get]
```

The minimum Y value that has been sampled.

Definition at line 77 of file Function2D.cs.

7.55.5.7 MinZ

```
double VectSharp.Plots.Function2DGrid.MinZ  [get]
```

The minimum value that has been obtained for the function.

Definition at line 87 of file Function2D.cs.

7.55.5.8 StepsX

```
int VectSharp.Plots.Function2DGrid.StepsX  [get]
```

If the function has been sampled along a regular grid, the number of steps between MinX and MaxX on the X axis.

Definition at line 98 of file Function2D.cs.

7.55.5.9 StepsY

```
int VectSharp.Plots.Function2DGrid.StepsY  [get]
```

If the function has been sampled along a regular grid, the number of steps between MinY and MaxY on the X axis.

Definition at line 104 of file Function2D.cs.

7.55.5.10 Type

```
GridType VectSharp.Plots.Function2DGrid.Type  [get]
```

The type of grid.

Definition at line 109 of file Function2D.cs.

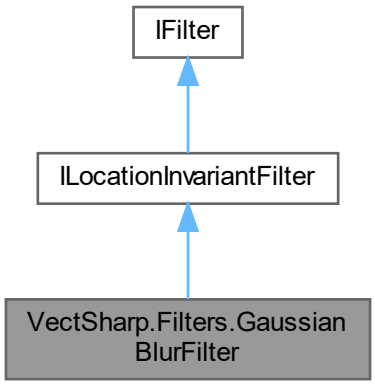
The documentation for this class was generated from the following file:

- VectSharp.Plots/Function2D.cs

7.56 VectSharp.Filters.GaussianBlurFilter Class Reference

Represents a filter that applies a Gaussian blur effect.

Inheritance diagram for VectSharp.Filters.GaussianBlurFilter:



Public Member Functions

- GaussianBlurFilter (double standardDeviation)

  Creates a new GaussianBlurFilter with the specified standard deviation.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- double StandardDeviation `[get]`

  The standard deviation of the Gaussian blur.
- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.56.1 Detailed Description

Represents a filter that applies a Gaussian blur effect.

Definition at line 26 of file GaussianBlurFilter.cs.

7.56.2 Constructor & Destructor Documentation

7.56.2.1 GaussianBlurFilter()

```
VectSharp.Filters.GaussianBlurFilter.GaussianBlurFilter (
            double standardDeviation )
```

Creates a new GaussianBlurFilter with the specified standard deviation.

Parameters

standardDeviation	The standard deviation of the Gaussian blur.

Definition at line 43 of file GaussianBlurFilter.cs.

7.56.3 Member Function Documentation

7.56.3.1 Filter()

```
RasterImage VectSharp.Filters.GaussianBlurFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 51 of file GaussianBlurFilter.cs.

7.56.4 Property Documentation

7.56.4.1 BottomRightMargin

```
Point VectSharp.Filters.GaussianBlurFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 37 of file GaussianBlurFilter.cs.

7.56.4.2 StandardDeviation

```
double VectSharp.Filters.GaussianBlurFilter.StandardDeviation  [get]
```

The standard deviation of the Gaussian blur.

Definition at line 31 of file GaussianBlurFilter.cs.

7.56.4.3 TopLeftMargin

```
Point VectSharp.Filters.GaussianBlurFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 34 of file GaussianBlurFilter.cs.

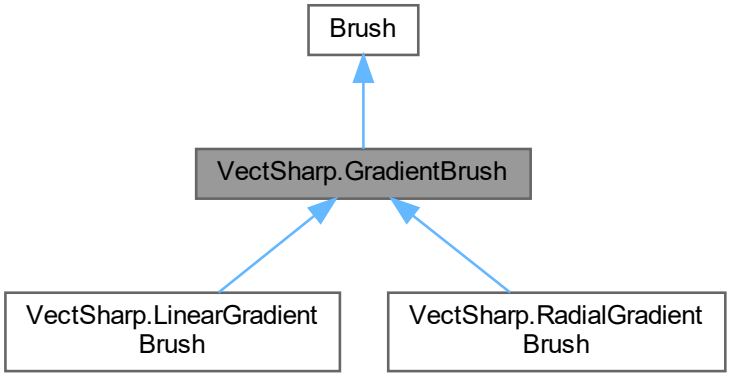
The documentation for this class was generated from the following file:

- VectSharp/Filters/GaussianBlurFilter.cs

7.57 VectSharp.GradientBrush Class Reference

Represents a brush painting with a gradient.

Inheritance diagram for VectSharp.GradientBrush:



Properties

- GradientStops GradientStops [get]

  The colour stops in the gradient.

Additional Inherited Members

7.57.1 Detailed Description

Represents a brush painting with a gradient.

Definition at line 270 of file Brush.cs.

7.57.2 Property Documentation

7.57.2.1 GradientStops

```
GradientStops VectSharp.GradientBrush.GradientStops [get]
```

The colour stops in the gradient.

Definition at line 275 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.58 VectSharp.Gradients Class Reference

Standard gradients.

Static Public Member Functions

- static Colour MagmaColouring (double x)

  Magma colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour InfernoColouring (double x)

  Inferno colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour PlasmaColouring (double x)

  Plasma colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour ViridisColouring (double x)

  Viridis colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour CividisColouring (double x)

  Cividis colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour RocketColouring (double x)

  Rocket colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour MakoColouring (double x)

  Mako colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour TurboColouring (double x)

  Turbo colour map, based on the homonymous colour scale included in the "viridis" R package ().

Static Public Attributes

- static readonly GradientStops TransparentToBlack = new GradientStops(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))

  Gradient from transparent black (0) to opaque black (1).
- static readonly GradientStops WhiteToBlack = new GradientStops(new GradientStop(Colour.FromRgb(255, 255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))

  Gradient from white (0) to black (1).
- static readonly GradientStops RedToGreen = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1))

  Gradient from red (0) to green (1).
- static readonly GradientStops Rainbow = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6), new GradientStop(Colour.FromRgb(255, 242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76), 0.5), new GradientStop(Colour.FromRgb(0, 162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72, 204), 5.0 / 6), new GradientStop(Colour.FromRgb(163, 73, 164), 1))

  Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).
- static readonly GradientStops RedYellowGreen = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new GradientStop(Colour.FromRgb(34, 177, 76), 1))

  Gradient from red (0) to yellow (0.5) to green (1).
- static readonly GradientStops OkabeItoRainbow = new GradientStops(new GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6), new GradientStop(Colour.FromRgb(230, 159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66), 0.5), new GradientStop(Colour.FromRgb(0, 158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114, 178), 5.0 / 6), new GradientStop(Colour.FromRgb(86, 180, 233), 1))

  Rainbow gradient with Okabe-Ito colour-blind safe colours ().

- static readonly GradientStops OkabeItoRainbowDiscrete = new GradientStops(new GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121, 167), 0.2), new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230, 159, 0), 0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0, 158, 115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0, 114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new GradientStop(Colour.FromRgb(86, 180, 233), 1))

  Rainbow gradient with Okabe-Ito colour-blind safe colours () in discrete steps.
- static readonly GradientStops MutedRainbow = new GradientStops(new GradientStop(Colour.FromRgb(221, 204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125), new GradientStop(Colour.FromRgb(17, 119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153), 0.375), new GradientStop(Colour.FromRgb(136, 204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34, 136), 0.625), new GradientStop(Colour.FromRgb(170, 68, 153), 0.75), new GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204, 102, 119), 1))

  Rainbow gradient with Paul Tol's Muted palette ().
- static readonly GradientStops MutedRainbowDiscrete = new GradientStops(new GradientStop(Colour.FromRgb(187, 187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187), 0.1), new GradientStop(Colour.FromRgb(221, 204, 119), 0.101), new GradientStop(Colour.FromRgb(221, 204, 119), 0.2), new GradientStop(Colour.FromRgb(153, 153, 51), 0.201), new GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17, 119, 51), 0.301), new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68, 170, 153), 0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136, 204, 238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new GradientStop(Colour.FromRgb(51, 34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7), new GradientStop(Colour.FromRgb(170, 68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153), 0.8), new GradientStop(Colour.FromRgb(136, 34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34, 85), 0.9), new GradientStop(Colour.FromRgb(204, 102, 119), 0.901), new GradientStop(Colour.FromRgb(204, 102, 119), 1))

  Rainbow gradient with Paul Tol's Muted palette () in discrete steps.
- static readonly GradientStops Magma = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(MagmaRGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el ∗ 0.05))

  Magma gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Inferno = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el ∗ 0.05))

  Inferno gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Plasma = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(PlasmaRGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el ∗ 0.05))

  Plasma gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Viridis = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el ∗ 0.05))

  Viridis gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Cividis = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el ∗ 0.05))

  Cividis gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Rocket = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(RocketRGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]), el ∗ 0.05))

  Rocket gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Mako = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(MakoRGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]), el ∗ 0.05))

  Mako gradient, based on the homonymous colour scale included in the "viridis" R package ().

- static readonly GradientStops Turbo = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(TurboRGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]), el ∗ 0.05))

  Turbo gradient, based on the homonymous colour scale included in the "viridis" R package ().

7.58.1 Detailed Description

Standard gradients.

Definition at line 13 of file Gradients.cs.

7.58.2 Member Function Documentation

7.58.2.1 CividisColouring()

```
static Colour VectSharp.Gradients.CividisColouring (
            double x )  [static]
```

Cividis colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 162 of file Gradients.cs.

7.58.2.2 InfernoColouring()

```
static Colour VectSharp.Gradients.InfernoColouring (
            double x )  [static]
```

Inferno colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 126 of file Gradients.cs.

7.58.2.3 MagmaColouring()

```
static Colour VectSharp.Gradients.MagmaColouring (
            double x )  [static]
```

Magma colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 114 of file Gradients.cs.

7.58.2.4 MakoColouring()

```
static Colour VectSharp.Gradients.MakoColouring (
            double x )  [static]
```

Mako colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 186 of file Gradients.cs.

7.58.2.5 PlasmaColouring()

```
static Colour VectSharp.Gradients.PlasmaColouring (
            double x )  [static]
```

Plasma colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 138 of file Gradients.cs.

7.58.2.6 RocketColouring()

```
static Colour VectSharp.Gradients.RocketColouring (
            double x )  [static]
```

Rocket colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 174 of file Gradients.cs.

7.58.2.7 TurboColouring()

```
static Colour VectSharp.Gradients.TurboColouring (
            double x )  [static]
```

Turbo colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 198 of file Gradients.cs.

7.58.2.8 ViridisColouring()

```
static Colour VectSharp.Gradients.ViridisColouring (
            double x )  [static]
```

Viridis colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 150 of file Gradients.cs.

7.58.3 Member Data Documentation

7.58.3.1 Cividis

```
readonly GradientStops VectSharp.Gradients.Cividis = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Cividis↩
RGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el * 0.05))  [static]
```

Cividis gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 92 of file Gradients.cs.

7.58.3.2 Inferno

```
readonly GradientStops VectSharp.Gradients.Inferno = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Inferno↩
RGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el * 0.05))  [static]
```

Inferno gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 77 of file Gradients.cs.

7.58.3.3 Magma

```
readonly GradientStops VectSharp.Gradients.Magma = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Magma↩
RGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el * 0.05))  [static]
```

Magma gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 72 of file Gradients.cs.

7.58.3.4 Mako

```
readonly GradientStops VectSharp.Gradients.Mako = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Mako↩
RGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]), el * 0.05))  [static]
```

Mako gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 102 of file Gradients.cs.

7.58.3.5 MutedRainbow

```
readonly GradientStops VectSharp.Gradients.MutedRainbow = new GradientStops(new GradientStop(Colour.FromRgb(22
204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125), new GradientStop(Colour.FromRgb(17,
119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153), 0.375), new GradientStop(Colour.FromRgb(136,
204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34, 136), 0.625), new GradientStop(Colour.FromRgb(170,
68, 153), 0.75), new GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204,
102, 119), 1))  [static]
```

Rainbow gradient with Paul Tol's Muted palette ().

Definition at line 53 of file Gradients.cs.

7.58.3.6 MutedRainbowDiscrete

```
readonly GradientStops VectSharp.Gradients.MutedRainbowDiscrete = new GradientStops(new GradientStop(Colour.Fr
187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187), 0.1), new GradientStop(Colour.FromRgb(221,
204, 119), 0.101), new GradientStop(Colour.FromRgb(221, 204, 119), 0.2), new GradientStop(Colour.FromRgb(153,
153, 51), 0.201), new GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17,
119, 51), 0.301), new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68,
170, 153), 0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136,
204, 238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new GradientStop(Colour.FromRgb(51,
34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7), new GradientStop(Colour.FromRgb(170,
68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153), 0.8), new GradientStop(Colour.FromRgb(136,
34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34, 85), 0.9), new GradientStop(Colour.FromRgb(204,
102, 119), 0.901), new GradientStop(Colour.FromRgb(204, 102, 119), 1))  [static]
```

Rainbow gradient with Paul Tol's Muted palette () in discrete steps.

Definition at line 58 of file Gradients.cs.

7.58.3.7 OkabeItoRainbow

```
readonly GradientStops VectSharp.Gradients.OkabeItoRainbow = new GradientStops(new GradientStop(Colour.FromRgb
121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6), new GradientStop(Colour.FromRgb(230,
159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66), 0.5), new GradientStop(Colour.FromRgb(0,
158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114, 178), 5.0 / 6), new GradientStop(Colour.FromRgb(8
180, 233), 1))  [static]
```

Rainbow gradient with Okabe-Ito colour-blind safe colours ().

Definition at line 43 of file Gradients.cs.

7.58.3.8 OkabeItoRainbowDiscrete

```
readonly GradientStops VectSharp.Gradients.OkabeItoRainbowDiscrete = new GradientStops(new
GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121,
167), 0.2), new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230,
159, 0), 0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0,
158, 115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0,
114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new GradientStop(Colour.FromRgb(86,
180, 233), 1))  [static]
```

Rainbow gradient with Okabe-Ito colour-blind safe colours () in discrete steps.

Definition at line 48 of file Gradients.cs.

7.58.3.9 Plasma

```
readonly GradientStops VectSharp.Gradients.Plasma = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Plasma↩
RGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el * 0.05))  [static]
```

Plasma gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 82 of file Gradients.cs.

7.58.3.10 Rainbow

```
readonly GradientStops VectSharp.Gradients.Rainbow = new GradientStops(new GradientStop(Colour.FromRgb(237,
28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6), new GradientStop(Colour.FromRgb(255,
242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76), 0.5), new GradientStop(Colour.FromRgb(0,
162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72, 204), 5.0 / 6), new GradientStop(Colour.FromRgb(1
73, 164), 1))  [static]
```

Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).

Definition at line 33 of file Gradients.cs.

7.58.3.11 RedToGreen

```
readonly GradientStops VectSharp.Gradients.RedToGreen = new GradientStops(new GradientStop(Colour.FromRgb(237,
28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1))  [static]
```

Gradient from red (0) to green (1).

Definition at line 28 of file Gradients.cs.

7.58.3.12 RedYellowGreen

```
readonly GradientStops VectSharp.Gradients.RedYellowGreen = new GradientStops(new GradientStop(Colour.FromRgb(
28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new GradientStop(Colour.FromRgb(34,
177, 76), 1))  [static]
```

Gradient from red (0) to yellow (0.5) to green (1).

Definition at line 38 of file Gradients.cs.

7.58.3.13 Rocket

```
readonly GradientStops VectSharp.Gradients.Rocket = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Rocket↩
RGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]), el * 0.05))  [static]
```

Rocket gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 97 of file Gradients.cs.

7.58.3.14 TransparentToBlack

```
readonly GradientStops VectSharp.Gradients.TransparentToBlack = new GradientStops(new GradientStop(Colour.From
0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))  [static]
```

Gradient from transparent black (0) to opaque black (1).

Definition at line 18 of file Gradients.cs.

7.58.3.15 Turbo

```
readonly GradientStops VectSharp.Gradients.Turbo = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Turbo↩
RGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]), el * 0.05))  [static]
```

Turbo gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 107 of file Gradients.cs.

7.58.3.16 Viridis

```
readonly GradientStops VectSharp.Gradients.Viridis = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Viridis↩
RGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el * 0.05))  [static]
```

Viridis gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 87 of file Gradients.cs.

7.58.3.17 WhiteToBlack

```
readonly GradientStops VectSharp.Gradients.WhiteToBlack = new GradientStops(new GradientStop(Colour.FromRgb(25
255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))  [static]
```

Gradient from white (0) to black (1).

Definition at line 23 of file Gradients.cs.

The documentation for this class was generated from the following file:

- VectSharp/Gradients.cs

7.59 VectSharp.GradientStop Struct Reference

Represents a colour stop in a gradient.

Public Member Functions

- GradientStop (Colour colour, double offset)

  Creates a new GradientStop instance.
- GradientStop MultiplyOpacity (double opacity)

  Returns a GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Properties

- Colour Colour [get]

  The Colour at the gradient stop.
- double Offset [get]

  The offset of the gradient stop. Range: [0, 1].

7.59.1 Detailed Description

Represents a colour stop in a gradient.

Definition at line 109 of file Brush.cs.

7.59.2 Constructor & Destructor Documentation

7.59.2.1 GradientStop()

```
VectSharp.GradientStop.GradientStop (
            Colour colour,
            double offset )
```

Creates a new GradientStop instance.

Parameters

colour	The Colour at the gradient stop.
offset	The offset of the gradient stop. Range: [0, 1].

Definition at line 126 of file Brush.cs.

7.59.3 Member Function Documentation

7.59.3.1 MultiplyOpacity()

```
GradientStop VectSharp.GradientStop.MultiplyOpacity (
            double opacity )
```

Returns a GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Parameters

opacity	The value that will be used to multiply the colour's opacity.

Returns

A GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Definition at line 137 of file Brush.cs.

7.59.4 Property Documentation

7.59.4.1 Colour

```
Colour VectSharp.GradientStop.Colour  [get]
```

The Colour at the gradient stop.

Definition at line 114 of file Brush.cs.

7.59.4.2 Offset

```
double VectSharp.GradientStop.Offset  [get]
```

The offset of the gradient stop. Range: [0, 1].

Definition at line 119 of file Brush.cs.

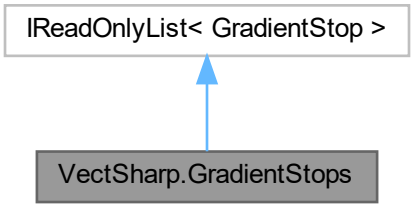
The documentation for this struct was generated from the following file:

- VectSharp/Brush.cs

7.60 VectSharp.GradientStops Class Reference

Represents a read-only list of GradientStops.

Inheritance diagram for VectSharp.GradientStops:



Public Member Functions

- IEnumerator< GradientStop > GetEnumerator ()
- GradientStops (IEnumerable< GradientStop > gradientStops)

  Creates a new GradientStops instance containing the specified gradient stops.
- GradientStops (params GradientStop[ ] gradientStops)

  Creates a new GradientStops instance containing the specified gradient stops.
- Colour GetColourAt (double position)

  Gets the colour at a certain position on the gradient.

Static Public Member Functions

- static implicit operator Func< double, Colour > (GradientStops stops)

  Convert a GradientStops object to a function that assigns a colour to values between 0 and 1.

Static Public Attributes

- static readonly double StopTolerance = 1e-7

  The minimum distance that is enforced between consecutive gradient stops.

Properties

- GradientStop this[int index] `[get]`
- int Count `[get]`

7.60.1 Detailed Description

Represents a read-only list of GradientStops.

Definition at line 146 of file Brush.cs.

7.60.2 Constructor & Destructor Documentation

7.60.2.1 GradientStops() `[1/2]`

```
VectSharp.GradientStops.GradientStops (
            IEnumerable< GradientStop > gradientStops )
```

Creates a new GradientStops instance containing the specified gradient stops.

Parameters

gradientStops	The gradient stops that will be contained in the GradientStops object.

Definition at line 176 of file Brush.cs.

7.60.2.2 GradientStops() [2/2]

```
VectSharp.GradientStops.GradientStops (
            params GradientStop[] gradientStops )
```

Creates a new GradientStops instance containing the specified gradient stops.

Parameters

gradientStops	The gradient stops that will be contained in the GradientStops object.

Definition at line 222 of file Brush.cs.

7.60.3 Member Function Documentation

7.60.3.1 GetColourAt()

```
Colour VectSharp.GradientStops.GetColourAt (
            double position )
```

Gets the colour at a certain position on the gradient.

Parameters

position	The position in the gradient (ranging from 0 to 1).

Returns

The colour of the gradient at the specified position.

Definition at line 232 of file Brush.cs.

7.60.3.2 GetEnumerator()

```
IEnumerator< GradientStop > VectSharp.GradientStops.GetEnumerator ( )
```

Definition at line 162 of file Brush.cs.

7.60.3.3 operator Func< double, Colour >()

```
static implicit VectSharp.GradientStops.operator Func< double, Colour > (
            GradientStops stops )  [static]
```

Convert a GradientStops object to a function that assigns a colour to values between 0 and 1.

Parameters

stops	The GradientStops to convert.

7.60.4 Member Data Documentation

7.60.4.1 StopTolerance

```
readonly double VectSharp.GradientStops.StopTolerance = 1e-7  [static]
```

The minimum distance that is enforced between consecutive gradient stops.

Definition at line 151 of file Brush.cs.

7.60.5 Property Documentation

7.60.5.1 Count

```
int VectSharp.GradientStops.Count  [get]
```

Definition at line 157 of file Brush.cs.

7.60.5.2 this[int index]

```
GradientStop VectSharp.GradientStops.this[int index]  [get]
```

Definition at line 154 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.61 VectSharp.Graphics Class Reference

Represents an abstract drawing surface.

Public Member Functions

- void FillPath (GraphicsPath path, Brush fillColour, string tag=null)

  Fill a GraphicsPath.
- void FillPath (GraphicsPath path, Brush fillColour, FillRule fillRule, string tag=null)

  Fill a GraphicsPath.
- void StrokePath (GraphicsPath path, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a GraphicsPath.
- void SetClippingPath (GraphicsPath path, string tag=null)

  Intersect the current clipping path with the specified GraphicsPath.
- void SetClippingPath (double leftX, double topY, double width, double height, string tag=null)

  Intersect the current clipping path with the specified rectangle.
- void SetClippingPath (Point topLeft, Size size, string tag=null)

  Intersect the current clipping path with the specified rectangle.
- void Rotate (double angle, string tag=null)

  Rotate the coordinate system around the origin.
- void RotateAt (double angle, Point pivot, string tag=null)

  Rotate the coordinate system around a pivot point.
- void Transform (double a, double b, double c, double d, double e, double f, string tag=null)

  Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].
- void Translate (double x, double y, string tag=null)

  Translate the coordinate system origin.
- void Translate (Point delta, string tag=null)

  Translate the coordinate system origin.
- void Scale (double scaleX, double scaleY, string tag=null)

  Scale the coordinate system with respect to the origin.
- void FillRectangle (Point topLeft, Size size, Brush fillColour, string tag=null)

  Fill a rectangle.
- void FillRectangle (double leftX, double topY, double width, double height, Brush fillColour, string tag=null)

  Fill a rectangle.
- void StrokeRectangle (Point topLeft, Size size, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a rectangle.
- void StrokeRectangle (double leftX, double topY, double width, double height, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a rectangle.
- void DrawRasterImage (int sourceX, int sourceY, int sourceWidth, int sourceHeight, double destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (double x, double y, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (Point position, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (double x, double y, double width, double height, RasterImage image, string tag=null)

Draw a raster image.

- void DrawRasterImage (Point position, Size size, RasterImage image, string tag=null)

  Draw a raster image.
- void Save ()

  Save the current transform state (rotation, translation, scale).
- void Restore ()

  Restore the previous transform state (rotation, translation scale).
- void CopyToIGraphicsContext (IGraphicsContext destinationContext)

  Copy the current graphics to an instance of a class implementing IGraphicsContext.
- void DrawGraphics (Point origin, Graphics graphics)

  Draws a Graphics object on the current Graphics object.
- void DrawGraphics (double originX, double originY, Graphics graphics, string tag)

  Draws a Graphics object on the current Graphics object, prepending the supplied tag to the tags contained in the Graphics object being drawn.
- void DrawGraphics (Point origin, Graphics graphics, string tag)

  Draws a Graphics object on the current Graphics object, prepending the supplied tag to the tags contained in the Graphics object being drawn.
- void DrawGraphics (double originX, double originY, Graphics graphics)

  Draws a Graphics object on the current Graphics object.
- void DrawGraphics (Point origin, Graphics graphics, IFilter filter, string tag=null)

  Draws a Graphics object on the current Graphics object, applying the specified filter .
- void DrawGraphics (double originX, double originY, Graphics graphics, IFilter filter, string tag=null)

  Draws a Graphics object on the current Graphics object, applying the specified filter .
- Graphics Transform (Func< Point, Point > transformationFunction, double linearisationResolution)

  Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.
- Graphics Transform (Func< Point, Point > transformationFunction, double linearisationResolution, double maxSegmentLength)

  Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.
- Graphics Linearise (double resolution)

  Creates a new Graphics object by linearising all of the elements of the current instance, i.e. replacing curve segments with series of line segments that approximate them. Raster images are left unchanged.
- Rectangle GetBounds ()

  Computes the rectangular bounds of the region affected by the drawing operations performed on the Graphics object.
- bool TryRasterise (Rectangle region, double scale, bool interpolate, out RasterImage output)

  Tries to rasterise specified region of this Graphics object using the default rasterisation method.
- void Crop (Rectangle region)

  Removes graphics actions that fall completely outside of the specified region .
- void Crop (Point topLeft, Size size)

  Removes graphics actions that fall completely outside of the specified region.
- IEnumerable< string > GetTags ()

  Gets all the tags that have been defined in the Graphics.
- void FillText (Point origin, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.↩ Top, string tag=null)

  Fill a text string.
- void FillText (double originX, double originY, string text, Font font, Brush fillColour, TextBaselines text↩ Baseline=TextBaselines.Top, string tag=null)

  Fill a text string.
- void StrokeText (Point origin, string text, Font font, Brush strokeColour, TextBaselines textBaseline=Text↩ Baselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

Stroke a text string.

- void StrokeText (double originX, double originY, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string.
- void FillTextOnPath (GraphicsPath path, string text, Font font, Brush fillColour, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a text string along a GraphicsPath.
- void StrokeTextOnPath (GraphicsPath path, string text, Font font, Brush strokeColour, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string along a GraphicsPath.
- void FillText (Point origin, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a formatted text string.
- void FillText (double originX, double originY, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a formatted text string.
- void StrokeText (Point origin, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a formatted text string.
- void StrokeText (double originX, double originY, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a formatted text string.
- Size MeasureText (string text, Font font)

  Measure a text string. See also

  See also

  Font.MeasureText(string), Font.MeasureTextAdvanced(string)

  and .
- Size MeasureText (IEnumerable< FormattedText > text)

  Measure a formatted text string. See also

  See also

  FormattedTextExtensions.Measure(IEnumerable<FormattedText>)

  .
- void FillTextUnderline (double originX, double originY, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fills the underline of the specified text string.
- void FillTextUnderline (Point origin, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fills the underline of the specified text string.
- void StrokeTextUnderline (double originX, double originY, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified text string.
- void StrokeTextUnderline (Point origin, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified text string.
- void FillTextUnderline (double originX, double originY, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

Fill the underline of the specified formatted text string.

- void FillTextUnderline (Point origin, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill the underline of the specified formatted text string.

- void StrokeTextUnderline (double originX, double originY, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified formatted text string.

- void StrokeTextUnderline (Point origin, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified formatted text string.

Static Public Attributes

- static Func< Graphics, Rectangle, double, bool, RasterImage > RasterisationMethod = null

  A method that is used to rasterise a region of a Graphics object. Set this to `null` if you wish to use the default rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp). You will have to provide your own implementation of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your project. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.

Properties

- static UnbalancedStackActions UnbalancedStackAction = UnbalancedStackActions.Throw `[get, set]`

  Determines how an unbalanced graphics state stack (which occurs if the number of calls to Save and Restore is not equal) will be treated. The default is UnbalancedStackActions.Throw.

- FillRule DefaultFillRule = FillRule.NonZeroWinding `[get, set]`

  The default fill rule.

- bool UseUniqueTags = true `[get, set]`

  Determines whether unique tags should be used for graphics actions that create multiple objects (e.g. drawing text).

7.61.1 Detailed Description

Represents an abstract drawing surface.

Definition at line 278 of file Graphics.cs.

7.61.2 Member Function Documentation

7.61.2.1 CopyToIGraphicsContext()

```
void VectSharp.Graphics.CopyToIGraphicsContext (
            IGraphicsContext destinationContext )
```

Copy the current graphics to an instance of a class implementing IGraphicsContext.

Parameters

destinationContext	The IGraphicsContext on which the graphics are to be copied.

Definition at line 666 of file Graphics.cs.

7.61.2.2 Crop() [1/2]

```
void VectSharp.Graphics.Crop (
            Point topLeft,
            Size size )
```

Removes graphics actions that fall completely outside of the specified region.

Parameters

topLeft	The top-left corner of the area to preserve.
size	The size of the area to preserve.

Definition at line 1695 of file Graphics.cs.

7.61.2.3 Crop() [2/2]

```
void VectSharp.Graphics.Crop (
            Rectangle region )
```

Removes graphics actions that fall completely outside of the specified *region* .

Parameters

region	The area to preserve.

Definition at line 1606 of file Graphics.cs.

7.61.2.4 DrawGraphics() [1/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics )
```

Draws a Graphics object on the current Graphics object.

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.

Definition at line 950 of file Graphics.cs.

7.61.2.5 DrawGraphics() [2/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics,
            IFilter filter,
            string tag = null )
```

Draws a Graphics object on the current Graphics object, applying the specified *filter* .

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.
tag	A tag to identify the filter.

Definition at line 992 of file Graphics.cs.

7.61.2.6 DrawGraphics() [3/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics,
            string tag )
```

Draws a Graphics object on the current Graphics object, prepending the supplied *tag* to the tags contained in the Graphics object being drawn.

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
tag	The tag to prepend to the tags contained in the *graphics* object.

Definition at line 893 of file Graphics.cs.

7.61.2.7 DrawGraphics() [4/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics )
```

Draws a Graphics object on the current Graphics object.

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.

Definition at line 874 of file Graphics.cs.

7.61.2.8 DrawGraphics() [5/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics,
            IFilter filter,
            string tag = null )
```

Draws a Graphics object on the current Graphics object, applying the specified *filter* .

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.
tag	A tag to identify the filter.

Definition at line 962 of file Graphics.cs.

7.61.2.9 DrawGraphics() [6/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics,
            string tag )
```

Draws a Graphics object on the current Graphics object, prepending the supplied *tag* to the tags contained in the Graphics object being drawn.

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
tag	The tag to prepend to the tags contained in the *graphics* object.

Definition at line 904 of file Graphics.cs.

7.61.2.10 DrawRasterImage() [1/5]

```
void VectSharp.Graphics.DrawRasterImage (
            double x,
            double y,
            double width,
            double height,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
y	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
width	The width of the rectangle delimiting the destination area of the image.
height	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 571 of file Graphics.cs.

7.61.2.11 DrawRasterImage() [2/5]

```
void VectSharp.Graphics.DrawRasterImage (
            double x,
            double y,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
y	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 546 of file Graphics.cs.

7.61.2.12 DrawRasterImage() [3/5]

```
void VectSharp.Graphics.DrawRasterImage (
            int sourceX,
            int sourceY,
            int sourceWidth,
            int sourceHeight,
            double destinationX,
            double destinationY,
            double destinationWidth,
            double destinationHeight,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

sourceX	The horizontal coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceY	The vertical coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceWidth	The width of the rectangle delimiting the source area of the image.
sourceHeight	The height of the rectangle delimiting the source area of the image.
destinationX	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationY	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationWidth	The width of the rectangle delimiting the destination area of the image.
destinationHeight	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 534 of file Graphics.cs.

7.61.2.13 DrawRasterImage() [4/5]

```
void VectSharp.Graphics.DrawRasterImage (
            Point position,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

position	The the top-left corner of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 557 of file Graphics.cs.

7.61.2.14 DrawRasterImage() [5/5]

```
void VectSharp.Graphics.DrawRasterImage (
            Point position,
            Size size,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

position	The the top-left corner of the rectangle delimiting the destination area of the image.
size	The size of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 583 of file Graphics.cs.

7.61.2.15 FillPath() [1/2]

```
void VectSharp.Graphics.FillPath (
            GraphicsPath path,
            Brush fillColour,
            FillRule fillRule,
            string tag = null )
```

Fill a GraphicsPath.

Parameters

path	The GraphicsPath to fill.
fillColour	The Brush with which to fill the GraphicsPath.
fillRule	The FillRule that determines which parts of the path are filled.
tag	A tag to identify the filled path.

Definition at line 310 of file Graphics.cs.

7.61.2.16 FillPath() [2/2]

```
void VectSharp.Graphics.FillPath (
            GraphicsPath path,
            Brush fillColour,
            string tag = null )
```

Fill a GraphicsPath.

Parameters

path	The GraphicsPath to fill.
fillColour	The Brush with which to fill the GraphicsPath.
tag	A tag to identify the filled path.

Definition at line 298 of file Graphics.cs.

7.61.2.17 FillRectangle() [1/2]

```
void VectSharp.Graphics.FillRectangle (
            double leftX,
            double topY,
            double width,
            double height,
            Brush fillColour,
            string tag = null )
```

Fill a rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
fillColour	The colour with which to fill the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 482 of file Graphics.cs.

7.61.2.18 FillRectangle() [2/2]

```
void VectSharp.Graphics.FillRectangle (
            Point topLeft,
            Size size,
            Brush fillColour,
            string tag = null )
```

Fill a rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
fillColour	The colour with which to fill the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 468 of file Graphics.cs.

7.61.2.19 FillText() [1/4]

```
void VectSharp.Graphics.FillText (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
fillColour	The default Brush to use to fill the text. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled text.

Definition at line 512 of file Graphics.Text.cs.

7.61.2.20 FillText() [2/4]

```
void VectSharp.Graphics.FillText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled text.

Definition at line 57 of file Graphics.Text.cs.

7.61.2.21 FillText() [3/4]

```
void VectSharp.Graphics.FillText (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
fillColour	The default Brush to use to fill the text. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled text.

Definition at line 411 of file Graphics.Text.cs.

7.61.2.22 FillText() [4/4]

```
void VectSharp.Graphics.FillText (
            Point origin,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled text.

Definition at line 34 of file Graphics.Text.cs.

7.61.2.23 FillTextOnPath()

```
void VectSharp.Graphics.FillTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
            Brush fillColour,
            double reference = 0,
            TextAnchors anchor = TextAnchors.Left,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string along a GraphicsPath.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .
tag	A tag to identify the filled text.

Definition at line 134 of file Graphics.Text.cs.

7.61.2.24 FillTextUnderline() [1/4]

```
void VectSharp.Graphics.FillTextUnderline (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill the underline of the specified formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText whose underline will be drawn.
fillColour	The default Brush to use to fill the underline. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled underlined.

Definition at line 754 of file Graphics.Text.cs.

7.61.2.25 FillTextUnderline() [2/4]

```
void VectSharp.Graphics.FillTextUnderline (
            double originX,
            double originY,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fills the underline of the specified text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string whose underline will be draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled underline.

Definition at line 680 of file Graphics.Text.cs.

7.61.2.26 FillTextUnderline() [3/4]

```
void VectSharp.Graphics.FillTextUnderline (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill the underline of the specified formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText whose underline will be drawn.
fillColour	The default Brush to use to fill the underline. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled underlined.

Definition at line 767 of file Graphics.Text.cs.

7.61.2.27 FillTextUnderline() [4/4]

```
void VectSharp.Graphics.FillTextUnderline (
            Point origin,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fills the underline of the specified text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled underline.

Definition at line 694 of file Graphics.Text.cs.

7.61.2.28 GetBounds()

```
Rectangle VectSharp.Graphics.GetBounds ( )
```

Computes the rectangular bounds of the region affected by the drawing operations performed on the Graphics object.

Returns

The smallest rectangle that contains all the elements drawn on the Graphics.

Definition at line 1453 of file Graphics.cs.

7.61.2.29 GetTags()

```
IEnumerable< string > VectSharp.Graphics.GetTags ( )
```

Gets all the tags that have been defined in the Graphics.

Returns

An IEnumerable<T> of strings that, when enumerated, returns all the tags that have been defined in the Graphics.

Definition at line 1705 of file Graphics.cs.

7.61.2.30 Linearise()

```
Graphics VectSharp.Graphics.Linearise (
            double resolution )
```

Creates a new Graphics object by linearising all of the elements of the current instance, i.e. replacing curve segments with series of line segments that approximate them. Raster images are left unchanged.

Parameters

resolution	The resolution that will be used to linearise curve segments.

Returns

A new Graphics object containing the linearised elements.

Definition at line 1397 of file Graphics.cs.

7.61.2.31 MeasureText() [1/2]

```
Size VectSharp.Graphics.MeasureText (
            IEnumerable< FormattedText > text )
```

Measure a formatted text string. See also

See also

FormattedTextExtensions.Measure(IEnumerable<FormattedText>)

.

Parameters

text	The collection of FormattedText objects to measure.

Returns

The size of the measured *text* .

Definition at line 663 of file Graphics.Text.cs.

7.61.2.32 MeasureText() [2/2]

```
Size VectSharp.Graphics.MeasureText (
            string text,
            Font font )
```

Measure a text string. See also

See also

Font.MeasureText(string), Font.MeasureTextAdvanced(string)

and .

Parameters

text	The string to measure.
font	The font to use to measure the string.

Returns

The size of the measured *text* .

Definition at line 645 of file Graphics.Text.cs.

7.61.2.33 Restore()

```
void VectSharp.Graphics.Restore ( )
```

Restore the previous transform state (rotation, translation scale).

Definition at line 599 of file Graphics.cs.

7.61.2.34 Rotate()

```
void VectSharp.Graphics.Rotate (
            double angle,
            string tag = null )
```

Rotate the coordinate system around the origin.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.
tag	A tag to identify the transform.

Definition at line 374 of file Graphics.cs.

7.61.2.35 RotateAt()

```
void VectSharp.Graphics.RotateAt (
            double angle,
            Point pivot,
            string tag = null )
```

Rotate the coordinate system around a pivot point.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.
pivot	The pivot around which the coordinate system is to be rotated.
tag	A tag to identify the transform.

Definition at line 385 of file Graphics.cs.

7.61.2.36 Save()

```
void VectSharp.Graphics.Save ( )
```

Save the current transform state (rotation, translation, scale).

Definition at line 591 of file Graphics.cs.

7.61.2.37 Scale()

```
void VectSharp.Graphics.Scale (
            double scaleX,
            double scaleY,
            string tag = null )
```

Scale the coordinate system with respect to the origin.

Parameters

scaleX	The horizontal scale.
scaleY	The vertical scale.
tag	A tag to identify the transform.

Definition at line 456 of file Graphics.cs.

7.61.2.38 SetClippingPath() [1/3]

```
void VectSharp.Graphics.SetClippingPath (
            double leftX,
            double topY,
            double width,
            double height,
            string tag = null )
```

Intersect the current clipping path with the specified rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
tag	A tag to identify the clipping path.

Definition at line 353 of file Graphics.cs.

7.61.2.39 SetClippingPath() [2/3]

```
void VectSharp.Graphics.SetClippingPath (
```

```
            GraphicsPath path,
            string tag = null )
```

Intersect the current clipping path with the specified GraphicsPath.

Parameters

path	The GraphicsPath to intersect with the current clipping path.
tag	A tag to identify the clipping path.

Definition at line 340 of file Graphics.cs.

7.61.2.40 SetClippingPath() [3/3]

```
void VectSharp.Graphics.SetClippingPath (
            Point topLeft,
            Size size,
            string tag = null )
```

Intersect the current clipping path with the specified rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
tag	A tag to identify the clipping path.

Definition at line 364 of file Graphics.cs.

7.61.2.41 StrokePath()

```
void VectSharp.Graphics.StrokePath (
            GraphicsPath path,
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a GraphicsPath.

Parameters

path	The GraphicsPath to stroke.
strokeColour	The Brush with which to stroke the GraphicsPath.
lineWidth	The width of the line with which the path is stroked.

Parameters

lineCap	The line cap to use to stroke the path.
lineJoin	The line join to use to stroke the path.
lineDash	The line dash to use to stroke the path.
tag	A tag to identify the stroked path.

Definition at line 330 of file Graphics.cs.

7.61.2.42 StrokeRectangle() [1/2]

```
void VectSharp.Graphics.StrokeRectangle (
            double leftX,
            double topY,
            double width,
            double height,
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
strokeColour	The colour with which to stroke the rectangle.
lineWidth	The width of the line with which the rectangle is stroked.
lineCap	The line cap to use to stroke the rectangle.
lineJoin	The line join to use to stroke the rectangle.
lineDash	The line dash to use to stroke the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 516 of file Graphics.cs.

7.61.2.43 StrokeRectangle() [2/2]

```
void VectSharp.Graphics.StrokeRectangle (
            Point topLeft,
            Size size,
```

```
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
strokeColour	The colour with which to stroke the rectangle.
lineWidth	The width of the line with which the rectangle is stroked.
lineCap	The line cap to use to stroke the rectangle.
lineJoin	The line join to use to stroke the rectangle.
lineDash	The line dash to use to stroke the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 498 of file Graphics.cs.

7.61.2.44 StrokeText() [1/4]

```
void VectSharp.Graphics.StrokeText (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked text.

Definition at line 633 of file Graphics.Text.cs.

7.61.2.45 StrokeText() [2/4]

```
void VectSharp.Graphics.StrokeText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked text.

Definition at line 110 of file Graphics.Text.cs.

7.61.2.46 StrokeText() [3/4]

```
void VectSharp.Graphics.StrokeText (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked text.

Definition at line 529 of file Graphics.Text.cs.

7.61.2.47 StrokeText() [4/4]

```
void VectSharp.Graphics.StrokeText (
            Point origin,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked text.

Definition at line 83 of file Graphics.Text.cs.

7.61.2.48 StrokeTextOnPath()

```
void VectSharp.Graphics.StrokeTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
            Brush strokeColour,
            double reference = 0,
            TextAnchors anchor = TextAnchors.Left,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string along a GraphicsPath.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .
tag	A tag to identify the stroked text.

Definition at line 276 of file Graphics.Text.cs.

7.61.2.49 StrokeTextUnderline() [1/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked underline.

Definition at line 874 of file Graphics.Text.cs.

7.61.2.50 StrokeTextUnderline() [2/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            double originX,
            double originY,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked underline.

Definition at line 717 of file Graphics.Text.cs.

7.61.2.51 StrokeTextUnderline() [3/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked underline.

Definition at line 891 of file Graphics.Text.cs.

7.61.2.52 StrokeTextUnderline() [4/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            Point origin,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the underline.

Parameters

lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked underline.

Definition at line 735 of file Graphics.Text.cs.

7.61.2.53 Transform() [1/3]

```
void VectSharp.Graphics.Transform (
            double a,
            double b,
            double c,
            double d,
            double e,
            double f,
            string tag = null )
```

Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].

Parameters

a	The first element of the first column.
b	The second element of the first column.
c	The first element of the second column.
d	The second element of the second column.
e	The first element of the third column.
f	The second element of the third column.
tag	A tag to identify the transform.

Definition at line 423 of file Graphics.cs.

7.61.2.54 Transform() [2/3]

```
Graphics VectSharp.Graphics.Transform (
            Func< Point, Point > transformationFunction,
            double linearisationResolution )
```

Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.

Parameters

transformationFunction	An arbitrary transformation function.
linearisationResolution	The resolution that will be used to linearise curve segments.

Returns

A new Graphics object in which all graphics actions have been linearised and transformed using the *transformationFunction* .

Definition at line 1070 of file Graphics.cs.

7.61.2.55 Transform() [3/3]

```
Graphics VectSharp.Graphics.Transform (
            Func< Point, Point > transformationFunction,
            double linearisationResolution,
            double maxSegmentLength )
```

Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.

Parameters

transformationFunction	An arbitrary transformation function.
linearisationResolution	The resolution that will be used to linearise curve segments.
maxSegmentLength	The maximum length of line segments.

Returns

A new Graphics object in which all graphics actions have been linearised and transformed using the *transformationFunction* .

Definition at line 1268 of file Graphics.cs.

7.61.2.56 Translate() [1/2]

```
void VectSharp.Graphics.Translate (
            double x,
            double y,
            string tag = null )
```

Translate the coordinate system origin.

Parameters

x	The horizontal translation.
y	The vertical translation.
tag	A tag to identify the transform.

Definition at line 435 of file Graphics.cs.

7.61.2.57 Translate() [2/2]

```
void VectSharp.Graphics.Translate (
            Point delta,
            string tag = null )
```

Translate the coordinate system origin.

Parameters

delta	The new origin point.
tag	A tag to identify the transform.

Definition at line 445 of file Graphics.cs.

7.61.2.58 TryRasterise()

```
bool VectSharp.Graphics.TryRasterise (
            Rectangle region,
            double scale,
            bool interpolate,
            out RasterImage output )
```

Tries to rasterise specified region of this Graphics object using the default rasterisation method.

Parameters

region	The region of the Graphics to rasterise.
scale	The scale at which the image is rasterised.
interpolate	Determines whether the resulting RasterImage should be interpolated or not.
output	When this method returns, this will contain the rasterised image (or `null` if the image could not be rasterised.

Returns

`true` if the image could be rasterised; `false` if it could not be rasterised.

Definition at line 1561 of file Graphics.cs.

7.61.3 Member Data Documentation

7.61.3.1 RasterisationMethod

```
Func<Graphics, Rectangle, double, bool, RasterImage> VectSharp.Graphics.RasterisationMethod =
null  [static]
```

A method that is used to rasterise a region of a Graphics object. Set this to `null` if you wish to use the default rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp). You will have to provide your own implementation of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your project. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.

Definition at line 1551 of file Graphics.cs.

7.61.4 Property Documentation

7.61.4.1 DefaultFillRule

```
FillRule VectSharp.Graphics.DefaultFillRule = FillRule.NonZeroWinding  [get], [set]
```

The default fill rule.

Definition at line 290 of file Graphics.cs.

7.61.4.2 UnbalancedStackAction

```
UnbalancedStackActions VectSharp.Graphics.UnbalancedStackAction = UnbalancedStackActions.Throw
[static], [get], [set]
```

Determines how an unbalanced graphics state stack (which occurs if the number of calls to Save and Restore is not equal) will be treated. The default is UnbalancedStackActions.Throw.

Definition at line 283 of file Graphics.cs.

7.61.4.3 UseUniqueTags

```
bool VectSharp.Graphics.UseUniqueTags = true  [get], [set]
```

Determines whether unique tags should be used for graphics actions that create multiple objects (e.g. drawing text).

Definition at line 318 of file Graphics.cs.

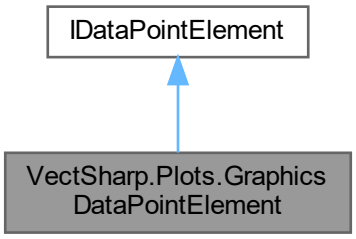
The documentation for this class was generated from the following files:

- VectSharp/Graphics.cs
- VectSharp/Graphics.Text.cs

7.62 VectSharp.Plots.GraphicsDataPointElement Class Reference

A symbol defined by a VectSharp.Graphics object.

Inheritance diagram for VectSharp.Plots.GraphicsDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- GraphicsDataPointElement (Graphics graphics)

  Creates a new GraphicsDataPointElement instance.

Properties

- Graphics Graphics `[get, set]`

  The VectSharp.Graphics object that will be copied on the plot.

7.62.1 Detailed Description

A symbol defined by a VectSharp.Graphics object.

Definition at line 96 of file DataPoints.cs.

7.62.2 Constructor & Destructor Documentation

7.62.2.1 GraphicsDataPointElement()

```
VectSharp.Plots.GraphicsDataPointElement.GraphicsDataPointElement (
            Graphics graphics )
```

Creates a new GraphicsDataPointElement instance.

Parameters

graphics	

Definition at line 113 of file DataPoints.cs.

7.62.3 Member Function Documentation

7.62.3.1 Plot()

```
void VectSharp.Plots.GraphicsDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

Definition at line 104 of file DataPoints.cs.

7.62.4 Property Documentation

7.62.4.1 Graphics

```
Graphics VectSharp.Plots.GraphicsDataPointElement.Graphics  [get], [set]
```

The VectSharp.Graphics object that will be copied on the plot.

Definition at line 101 of file DataPoints.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.63 VectSharp.GraphicsPath Class Reference

Represents a graphics path that can be filled or stroked.

Public Member Functions

- GraphicsPath MoveTo (Point p)

  Move the current point without tracing a segment from the previous point.
- GraphicsPath MoveTo (double x, double y)

  Move the current point without tracing a segment from the previous point.
- GraphicsPath LineTo (Point p)

  Move the current point and trace a segment from the previous point.
- GraphicsPath LineTo (double x, double y)

  Move the current point and trace a segment from the previous point.
- GraphicsPath Arc (Point center, double radius, double startAngle, double endAngle)

  Trace an arc segment from a circle with the specified center and radius , starting at startAngle and ending at endAngle . The current point is updated to the end point of the arc.
- GraphicsPath Arc (double centerX, double centerY, double radius, double startAngle, double endAngle)

  Trace an arc segment from a circle with the specified center and radius , starting at startAngle and ending at endAngle . The current point is updated to the end point of the arc.
- GraphicsPath EllipticalArc (double radiusX, double radiusY, double axisAngle, bool largeArc, bool sweep↩ Clockwise, Point endPoint)

  Trace an arc from an ellipse with the specified radii, rotated by axisAngle with respect to the x-axis, starting at the current point and ending at the endPoint .
- GraphicsPath CubicBezierTo (Point control1, Point control2, Point endPoint)

  Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.

- GraphicsPath CubicBezierTo (double control1X, double control1Y, double control2X, double control2Y, double endPointX, double endPointY)

  Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.
- GraphicsPath QuadraticBezierTo (Point control, Point endPoint)

  Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.
- GraphicsPath QuadraticBezierTo (double controlX, double controlY, double endPointX, double endPointY)

  Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.
- GraphicsPath Close ()

  Trace a segment from the current point to the start point of the figure and flag the figure as closed.
- GraphicsPath AddText (double originX, double originY, string text, Font font, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of a text string to the current path.
- GraphicsPath AddText (Point origin, string text, Font font, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of a text string to the current path.
- GraphicsPath AddTextOnPath (GraphicsPath path, string text, Font font, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of a text string flowing along a GraphicsPath to the current path.
- GraphicsPath AddTextUnderline (Point origin, string text, Font font, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of the underline of the specified text string to the current path.
- GraphicsPath AddSmoothSpline (params Point[ ] points)

  Adds a smooth spline composed of cubic bezier segments that pass through the specified points.
- GraphicsPath AddPath (GraphicsPath path)

  Adds another GraphicsPath to the current GraphicsPath.
- double MeasureLength ()

  Measures the length of the GraphicsPath.
- Point GetPointAtRelative (double position)

  Gets the point at the relative position specified on the GraphicsPath.
- Point GetPointAtAbsolute (double length)

  Gets the point at the absolute position specified on the GraphicsPath.
- Point GetTangentAtRelative (double position)

  Gets the tangent to the point at the relative position specified on the GraphicsPath.
- Point GetTangentAtAbsolute (double length)

  Gets the tangent to the point at the absolute position specified on the GraphicsPath.
- Point GetNormalAtAbsolute (double length)

  Gets the normal to the point at the absolute position specified on the GraphicsPath.
- Point GetNormalAtRelative (double position)

  Gets the normal to the point at the relative position specified on the GraphicsPath.
- GraphicsPath Linearise (double resolution)

  Linearises a GraphicsPath, replacing curve segments with series of line segments that approximate them.
- GraphicsPath Discretise (double resolution)

  Discretises a GraphicsPath, replacing curve segments with series of line segments that approximate them and ensuring that all line segments are shorter than the specified resolution .
- GraphicsPath Flatten (double flatness)

  Flattens a GraphicsPath, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.
- IEnumerable< List< Point > > GetPoints ()

  Gets a collection of the end points of all the segments in the GraphicsPath, divided by figure.
- IEnumerable< GraphicsPath > GetFigures ()

Gets a collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.

- IEnumerable< List< Point > > GetLinearisationPointsNormals (double resolution)

  Gets a collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.
- IEnumerable< GraphicsPath > Triangulate (double resolution, bool clockwise)

  Divides a GraphicsPath into triangles.
- GraphicsPath Transform (Func< Point, Point > transformationFunction)

  Transforms all of the Points in the GraphicsPath with an arbitrary transformation function.
- Rectangle GetBounds ()

  Compute the rectangular bounds of the path.
- GraphicsPath Reverse ()

  Reverses the GraphicsPath.
- GraphicsPath GetStroke (double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter)

  Returns a GraphicsPath representing the stroke of the current GraphicsPath.
- bool ContainsPoint (Point point, FillRule fillRule)

  Determines whether the specified point falls within the current GraphicsPath.

Properties

- List< Segment > Segments = new List() `[get, set]`

  The segments that make up the path.

7.63.1 Detailed Description

Represents a graphics path that can be filled or stroked.

Definition at line 27 of file GraphicsPath.cs.

7.63.2 Member Function Documentation

7.63.2.1 AddPath()

```
GraphicsPath VectSharp.GraphicsPath.AddPath (
            GraphicsPath path )
```

Adds another GraphicsPath to the current GraphicsPath.

Parameters

path	The existing GraphicsPath that should be added to the current GraphicsPath.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 1209 of file GraphicsPath.cs.

7.63.2.2 AddSmoothSpline()

```
GraphicsPath VectSharp.GraphicsPath.AddSmoothSpline (
            params Point[] points )
```

Adds a smooth spline composed of cubic bezier segments that pass through the specified points.

Parameters

points	The points through which the spline should pass.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 1177 of file GraphicsPath.cs.

7.63.2.3 AddText() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.AddText (
            double originX,
            double originY,
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string to the current path.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what *originY* represents).

///

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 357 of file GraphicsPath.cs.

7.63.2.4 AddText() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.AddText (
            Point origin,
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string to the current path.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 370 of file GraphicsPath.cs.

7.63.2.5 AddTextOnPath()

```
GraphicsPath VectSharp.GraphicsPath.AddTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
            double reference = 0,
            TextAnchors anchor = TextAnchors.Left,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string flowing along a GraphicsPath to the current path.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 475 of file GraphicsPath.cs.

7.63.2.6 AddTextUnderline()

```
GraphicsPath VectSharp.GraphicsPath.AddTextUnderline (
            Point origin,
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of the underline of the specified text string to the current path.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 623 of file GraphicsPath.cs.

7.63.2.7 Arc() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.Arc (
            double centerX,
            double centerY,
            double radius,
            double startAngle,
            double endAngle )
```

Trace an arc segment from a circle with the specified center and *radius* , starting at *startAngle* and ending at *endAngle* . The current point is updated to the end point of the arc.

Parameters

centerX	The horizontal coordinate of the center of the arc.
centerY	The vertical coordinate of the center of the arc.
radius	The radius of the arc.
startAngle	The start angle (in radians) of the arc.
endAngle	The end angle (in radians) of the arc.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 125 of file GraphicsPath.cs.

7.63.2.8 Arc() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.Arc (
            Point center,
            double radius,
            double startAngle,
            double endAngle )
```

Trace an arc segment from a circle with the specified *center* and *radius* , starting at *startAngle* and ending at *endAngle* . The current point is updated to the end point of the arc.

Parameters

center	The center of the arc.
radius	The radius of the arc.
startAngle	The start angle (in radians) of the arc.
endAngle	The end angle (in radians) of the arc.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 102 of file GraphicsPath.cs.

7.63.2.9 Close()

```
GraphicsPath VectSharp.GraphicsPath.Close ( )
```

Trace a segment from the current point to the start point of the figure and flag the figure as closed.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 340 of file GraphicsPath.cs.

7.63.2.10 ContainsPoint()

```
bool VectSharp.GraphicsPath.ContainsPoint (
            Point point,
            FillRule fillRule )
```

Determines whether the specified *point* falls within the current GraphicsPath.

Parameters

point	The Point being analysed.
fillRule	The rule to use to determine whether a point is inside or outside of a the GraphicsPath.

Returns

`true` if the *point* is inside the GraphicsPath, or `false` if it is outside. If the point lies exactly on the GraphicsPath (or very close to it), the result may be either `true` or `false`, depending on numerical approximations.

Definition at line 3591 of file GraphicsPath.cs.

7.63.2.11 CubicBezierTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.CubicBezierTo (
            double control1X,
            double control1Y,
            double control2X,
            double control2Y,
            double endPointX,
            double endPointY )
```

Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.

Parameters

control1X	The horizontal coordinate of the first control point.
control1Y	The vertical coordinate of the first control point.
control2X	The horizontal coordinate of the second control point.
control2Y	The vertical coordinate of the second control point.
endPointX	The horizontal coordinate of the destination point.
endPointY	The vertical coordinate of the destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 284 of file GraphicsPath.cs.

7.63.2.12 CubicBezierTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.CubicBezierTo (
            Point control1,
```

```
            Point control2,
            Point endPoint )
```

Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.

Parameters

control1	The first control point.
control2	The second control point.
endPoint	The destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 260 of file GraphicsPath.cs.

7.63.2.13 Discretise()

```
GraphicsPath VectSharp.GraphicsPath.Discretise (
            double resolution )
```

Discretises a GraphicsPath, replacing curve segments with series of line segments that approximate them and ensuring that all line segments are shorter than the specified *resolution* .

Parameters

resolution	The maximum length (in absolute units) of line segments in the resulting GraphicsPath.

Returns

A GraphicsPath composed only of linear segments that are shorter than *resolution* and approximate the current GraphicsPath.

Definition at line 1963 of file GraphicsPath.cs.

7.63.2.14 EllipticalArc()

```
GraphicsPath VectSharp.GraphicsPath.EllipticalArc (
            double radiusX,
            double radiusY,
            double axisAngle,
            bool largeArc,
            bool sweepClockwise,
            Point endPoint )
```

Trace an arc from an ellipse with the specified radii, rotated by *axisAngle* with respect to the x-axis, starting at the current point and ending at the *endPoint* .

Parameters

radiusX	The horizontal radius of the ellipse.
radiusY	The vertical radius of the ellipse.
axisAngle	The angle of the horizontal axis of the ellipse with respect to the horizontal axis.
largeArc	Determines whether the large or the small arc is drawn.
sweepClockwise	Determines whether the clockwise or anticlockwise arc is drawn.
endPoint	The end point of the arc.

Returns

Definition at line 141 of file GraphicsPath.cs.

7.63.2.15 Flatten()

```
GraphicsPath VectSharp.GraphicsPath.Flatten (
            double flatness )
```

Flattens a GraphicsPath, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.

Parameters

flatness	The maximum deviation from the original path.

Returns

A GraphicsPath composed only of linear segments that approximates the current GraphicsPath.

Definition at line 1993 of file GraphicsPath.cs.

7.63.2.16 GetBounds()

```
Rectangle VectSharp.GraphicsPath.GetBounds ( )
```

Compute the rectangular bounds of the path.

Returns

The smallest Rectangle that contains the path.

Definition at line 3031 of file GraphicsPath.cs.

7.63.2.17 GetFigures()

```
IEnumerable< GraphicsPath > VectSharp.GraphicsPath.GetFigures ( )
```

Gets a collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.

Returns

A collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.

Definition at line 2089 of file GraphicsPath.cs.

7.63.2.18 GetLinearisationPointsNormals()

```
IEnumerable< List< Point > > VectSharp.GraphicsPath.GetLinearisationPointsNormals (
            double resolution )
```

Gets a collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.

Parameters

resolution	The absolute length between successive samples in curve segments.

Returns

A collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.

Definition at line 2132 of file GraphicsPath.cs.

7.63.2.19 GetNormalAtAbsolute()

```
Point VectSharp.GraphicsPath.GetNormalAtAbsolute (
            double length )
```

Gets the normal to the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The normal to the point at the specified position.

Definition at line 1887 of file GraphicsPath.cs.

7.63.2.20 GetNormalAtRelative()

```
Point VectSharp.GraphicsPath.GetNormalAtRelative (
            double position )
```

Gets the normal to the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The normal to the point at the specified position.

Definition at line 1898 of file GraphicsPath.cs.

7.63.2.21 GetPointAtAbsolute()

```
Point VectSharp.GraphicsPath.GetPointAtAbsolute (
            double length )
```

Gets the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The point at the specified position.

Definition at line 1303 of file GraphicsPath.cs.

7.63.2.22 GetPointAtRelative()

```
Point VectSharp.GraphicsPath.GetPointAtRelative (
            double position )
```

Gets the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The point at the specified position.

Definition at line 1293 of file GraphicsPath.cs.

7.63.2.23 GetPoints()

```
IEnumerable< List< Point > > VectSharp.GraphicsPath.GetPoints ( )
```

Gets a collection of the end points of all the segments in the GraphicsPath, divided by figure.

Returns

A collection of the end points of all the segments in the GraphicsPath, divided by figure.

Definition at line 2046 of file GraphicsPath.cs.

7.63.2.24 GetStroke()

```
GraphicsPath VectSharp.GraphicsPath.GetStroke (
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter )
```

Returns a GraphicsPath representing the stroke of the current GraphicsPath.

Parameters

lineWidth	The thickness of the stroke.
lineCap	The line cap used in the stroke.
lineJoin	The line join used in the stroke.

Returns

A GraphicsPath representing the stroke of the current GraphicsPath.

Definition at line 3437 of file GraphicsPath.cs.

7.63.2.25 GetTangentAtAbsolute()

```
Point VectSharp.GraphicsPath.GetTangentAtAbsolute (
            double length )
```

Gets the tangent to the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The tangent to the point at the specified position.

Definition at line 1600 of file GraphicsPath.cs.

7.63.2.26 GetTangentAtRelative()

```
Point VectSharp.GraphicsPath.GetTangentAtRelative (
            double position )
```

Gets the tangent to the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The tangent to the point at the specified position.

Definition at line 1590 of file GraphicsPath.cs.

7.63.2.27 Linearise()

```
GraphicsPath VectSharp.GraphicsPath.Linearise (
            double resolution )
```

Linearises a GraphicsPath, replacing curve segments with series of line segments that approximate them.

Parameters

resolution	The absolute length between successive samples in curve segments.

Returns

A GraphicsPath composed only of linear segments that approximates the current GraphicsPath.

Definition at line 1909 of file GraphicsPath.cs.

7.63.2.28 LineTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.LineTo (
            double x,
            double y )
```

Move the current point and trace a segment from the previous point.

Parameters

x	The horizontal coordinate of the new point.
y	The vertical coordinate of the new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 87 of file GraphicsPath.cs.

7.63.2.29 LineTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.LineTo (
            Point p )
```

Move the current point and trace a segment from the previous point.

Parameters

p	The new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 65 of file GraphicsPath.cs.

7.63.2.30 MeasureLength()

```
double VectSharp.GraphicsPath.MeasureLength ( )
```

Measures the length of the GraphicsPath.

Returns

The length of the GraphicsPath

Definition at line 1221 of file GraphicsPath.cs.

7.63.2.31 MoveTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.MoveTo (
            double x,
            double y )
```

Move the current point without tracing a segment from the previous point.

Parameters

x	The horizontal coordinate of the new point.
y	The vertical coordinate of the new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 54 of file GraphicsPath.cs.

7.63.2.32 MoveTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.MoveTo (
            Point p )
```

Move the current point without tracing a segment from the previous point.

Parameters

p	The new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 40 of file GraphicsPath.cs.

7.63.2.33 QuadraticBezierTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.QuadraticBezierTo (
            double controlX,
            double controlY,
            double endPointX,
            double endPointY )
```

Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.

Parameters

controlX	The horizontal coordinate of the control point.
controlY	The vertical coordinate of the control point.
endPointX	The horizontal coordinate of the destination point.
endPointY	The vertical coordinate of the destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 330 of file GraphicsPath.cs.

7.63.2.34 QuadraticBezierTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.QuadraticBezierTo (
            Point control,
            Point endPoint )
```

Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.

Parameters

control	The control point.
endPoint	The destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 297 of file GraphicsPath.cs.

7.63.2.35 Reverse()

```
GraphicsPath VectSharp.GraphicsPath.Reverse ( )
```

Reverses the GraphicsPath.

Returns

The reversed GraphicsPath.

Definition at line 3328 of file GraphicsPath.cs.

7.63.2.36 Transform()

```
GraphicsPath VectSharp.GraphicsPath.Transform (
            Func< Point, Point > transformationFunction )
```

Transforms all of the Points in the GraphicsPath with an arbitrary transformation function.

Parameters

transformationFunction	An arbitrary transformation function.

Returns

A new GraphicsPath in which all points have been replaced using the *transformationFunction* .

Definition at line 3012 of file GraphicsPath.cs.

7.63.2.37 Triangulate()

```
IEnumerable< GraphicsPath > VectSharp.GraphicsPath.Triangulate (
            double resolution,
            bool clockwise )
```

Divides a GraphicsPath into triangles.

Parameters

resolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
clockwise	If this is `true`, the triangles will have their vertices in a clockwise order, otherwise they will be in anticlockwise order.

Returns

A collection of distinct GraphicsPaths, each representing one triangle.

Definition at line 2225 of file GraphicsPath.cs.

7.63.3 Property Documentation

7.63.3.1 Segments

```
List<Segment> VectSharp.GraphicsPath.Segments = new List<Segment>()  [get], [set]
```

The segments that make up the path.

Definition at line 32 of file GraphicsPath.cs.

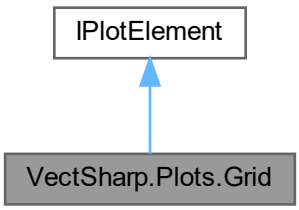
The documentation for this class was generated from the following file:

- VectSharp/GraphicsPath.cs

7.64 VectSharp.Plots.Grid Class Reference

A plot element that draws a grid.

Inheritance diagram for VectSharp.Plots.Grid:



Public Member Functions

- Grid (IReadOnlyList< double > side1Start, IReadOnlyList< double > side1End, IReadOnlyList< double > side2Start, IReadOnlyList< double > side2End, IContinuousCoordinateSystem coordinateSystem)

  Create a new Grid instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Side1Start `[get, set]`

  The starting point for the first side of the grid.
- IReadOnlyList< double > Side1End `[get, set]`

  The ending point for the first side of the grid.
- IReadOnlyList< double > Side2Start `[get, set]`

  The starting point for the second side of the grid.
- IReadOnlyList< double > Side2End `[get, set]`

  The ending point for the second side of the grid.
- int IntervalCount = 10 `[get, set]`

  The number of intervals between grid lines. Note that the number of grid lines will be one greater than this.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) } `[get, set]`

  Presentation attributes determining the appearance of the grid.
- string Tag `[get, set]`

  A tag to identify the grid in the plot.

7.64.1 Detailed Description

A plot element that draws a grid.

Definition at line 827 of file Axes.cs.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 Grid()

```
VectSharp.Plots.Grid.Grid (
            IReadOnlyList< double > side1Start,
            IReadOnlyList< double > side1End,
            IReadOnlyList< double > side2Start,
            IReadOnlyList< double > side2End,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new Grid instance.

Parameters

side1Start	The starting point for the first side of the grid.
side1End	The ending point for the first side of the grid.
side2Start	The starting point for the second side of the grid.
side2End	The ending point for the second side of the grid.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 878 of file Axes.cs.

7.64.3 Member Function Documentation

7.64.3.1 Plot()

```
void VectSharp.Plots.Grid.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 888 of file Axes.cs.

7.64.4 Property Documentation

7.64.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.Grid.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 857 of file Axes.cs.

7.64.4.2 IntervalCount

```
int VectSharp.Plots.Grid.IntervalCount = 10  [get], [set]
```

The number of intervals between grid lines. Note that the number of grid lines will be one greater than this.

Definition at line 852 of file Axes.cs.

7.64.4.3 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Grid.PresentationAttributes = new PlotElementPresentationAtt
{ Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) }  [get], [set]
```

Presentation attributes determining the appearance of the grid.

Definition at line 863 of file Axes.cs.

7.64.4.4 Side1End

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side1End  [get], [set]
```

The ending point for the first side of the grid.

Definition at line 837 of file Axes.cs.

7.64.4.5 Side1Start

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side1Start  [get], [set]
```

The starting point for the first side of the grid.

Definition at line 832 of file Axes.cs.

7.64.4.6 Side2End

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side2End  [get], [set]
```

The ending point for the second side of the grid.

Definition at line 847 of file Axes.cs.

7.64.4.7 Side2Start

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side2Start  [get], [set]
```

The starting point for the second side of the grid.

Definition at line 842 of file Axes.cs.

7.64.4.8 Tag

```
string VectSharp.Plots.Grid.Tag  [get], [set]
```

A tag to identify the grid in the plot.

Definition at line 868 of file Axes.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.65 VectSharp.Markdown.HTTPUtils Class Reference

Contains utilities to resolve absolute and relative URIs.

Static Public Member Functions

- static string bool wasDownloaded ResolveImageURI (string uri, string baseUriString)

Static Public Attributes

- static string path

  Resolves an image Uri, by downloading the image file if necessary. It also takes care of ensuring that the file extension matches the format of the file.

Properties

- static bool LogDownloads = true `[get, set]`

  Determines whether every file that is downloaded should be logged to the standard error stream.

7.65.1 Detailed Description

Contains utilities to resolve absolute and relative URIs.

Definition at line 244 of file HtmlTag.cs.

7.65.2 Member Function Documentation

7.65.2.1 ResolveImageURI()

```
static string bool wasDownloaded VectSharp.Markdown.HTTPUtils.ResolveImageURI (
            string uri,
            string baseUriString )  [static]
```

Definition at line 257 of file HtmlTag.cs.

7.65.3 Member Data Documentation

7.65.3.1 path

```
string VectSharp.Markdown.HTTPUtils.path  [static]
```

Resolves an image Uri, by downloading the image file if necessary. It also takes care of ensuring that the file extension matches the format of the file.

Parameters

uri	The address of the image.
baseUriString	The base uri to use for relative uris.

Returns

A tuple containing the local path of the image file (either the original image, or a local copy of a remote file) and a boolean value indicating whether the image was fetched from a remote location and should be deleted after the program is done with it.

Definition at line 257 of file HtmlTag.cs.

7.65.4 Property Documentation

7.65.4.1 LogDownloads

```
bool VectSharp.Markdown.HTTPUtils.LogDownloads = true  [static], [get], [set]
```

Determines whether every file that is downloaded should be logged to the standard error stream.

Definition at line 249 of file HtmlTag.cs.

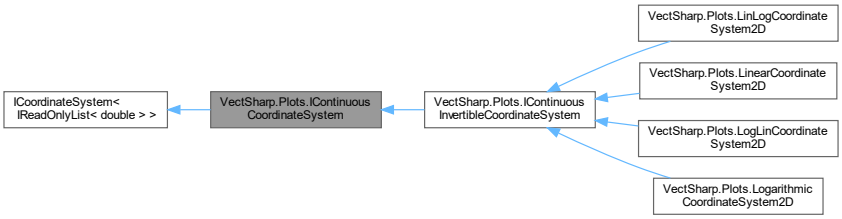
The documentation for this class was generated from the following file:

- VectSharp.Markdown/HtmlTag.cs

7.66 VectSharp.Plots.IContinuousCoordinateSystem Interface Reference

Represents a coordinate system performing continuous transformations.

Inheritance diagram for VectSharp.Plots.IContinuousCoordinateSystem:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.66.1 Detailed Description

Represents a coordinate system performing continuous transformations.

Definition at line 89 of file CoordinateSystems.cs.

7.66.2 Member Function Documentation

7.66.2.1 GetAround()

```
double[] VectSharp.Plots.IContinuousCoordinateSystem.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

7.66.2.2 IsDirectionStraight()

```
bool VectSharp.Plots.IContinuousCoordinateSystem.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

7.66.3 Property Documentation

7.66.3.1 IsLinear

```
bool VectSharp.Plots.IContinuousCoordinateSystem.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

Definition at line 94 of file CoordinateSystems.cs.

7.66.3.2 Resolution

```
double [] VectSharp.Plots.IContinuousCoordinateSystem.Resolution  [get]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

Definition at line 106 of file CoordinateSystems.cs.

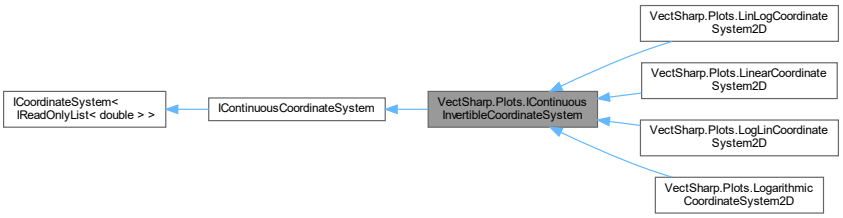
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.67 VectSharp.Plots.IContinuousInvertibleCoordinateSystem Interface Reference

Represents a coordinate system performing continuous invertible transformations.

Inheritance diagram for VectSharp.Plots.IContinuousInvertibleCoordinateSystem:



Public Member Functions

- double[] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Additional Inherited Members

7.67.1 Detailed Description

Represents a coordinate system performing continuous invertible transformations.

Definition at line 120 of file CoordinateSystems.cs.

7.67.2 Member Function Documentation

7.67.2.1 ToDataCoordinates()

```
double[] VectSharp.Plots.IContinuousInvertibleCoordinateSystem.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

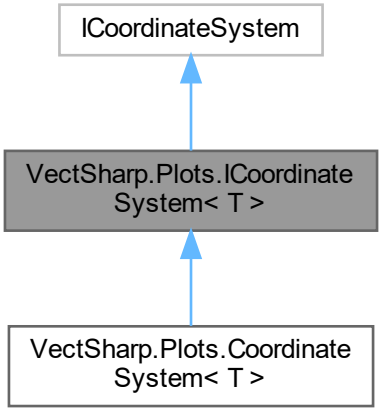
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.68 VectSharp.Plots.ICoordinateSystem< T > Interface Template Reference

Represents a coordinate system.

Inheritance diagram for VectSharp.Plots.ICoordinateSystem< T >:



Public Member Functions

- Point ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a plot Point.

7.68.1 Detailed Description

Represents a coordinate system.

Represents a coordinate system transforming data points of type *T* into plot Points.

Template Parameters

T	The type of data elements.

Definition at line 35 of file CoordinateSystems.cs.

7.68.2 Member Function Documentation

7.68.2.1 ToPlotCoordinates()

```
Point VectSharp.Plots.ICoordinateSystem< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a plot Point.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Implemented in VectSharp.Plots.CoordinateSystem< T >.

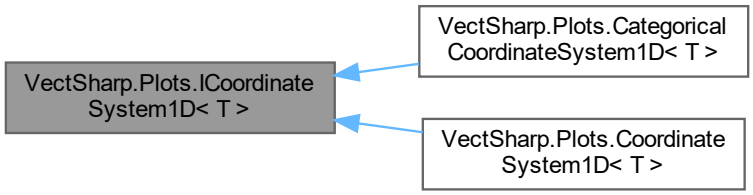
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.69 VectSharp.Plots.ICoordinateSystem1D< T > Interface Template Reference

Represents a coordinate system tranforming data points of type *T* into `double`s.

Inheritance diagram for VectSharp.Plots.ICoordinateSystem1D< T >:



Public Member Functions

- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`.

7.69.1 Detailed Description

Represents a coordinate system tranforming data points of type *T* into `double`s.

Template Parameters

T	The type of data elements.

Definition at line 49 of file CoordinateSystems.cs.

7.69.2 Member Function Documentation

7.69.2.1 ToPlotCoordinates()

```
double VectSharp.Plots.ICoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implemented in VectSharp.Plots.CoordinateSystem1D< T >, and VectSharp.Plots.CategoricalCoordinateSystem1D< T >.

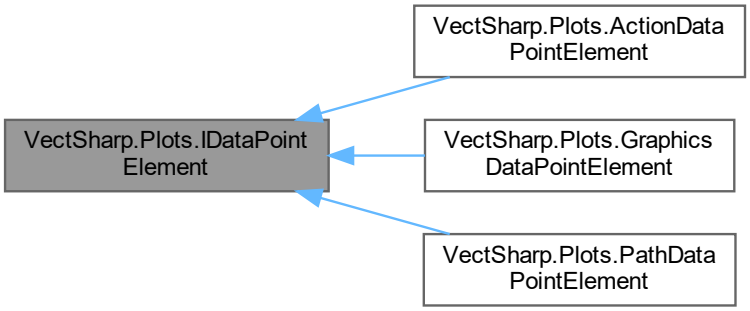
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.70 VectSharp.Plots.IDataPointElement Interface Reference

Represents a symbol that can be added to the plot at a specified position.

Inheritance diagram for VectSharp.Plots.IDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

7.70.1 Detailed Description

Represents a symbol that can be added to the plot at a specified position.

Definition at line 26 of file DataPoints.cs.

7.70.2 Member Function Documentation

7.70.2.1 Plot()

```
void VectSharp.Plots.IDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implemented in VectSharp.Plots.PathDataPointElement, VectSharp.Plots.GraphicsDataPointElement, and VectSharp.Plots.ActionDataPointElement.

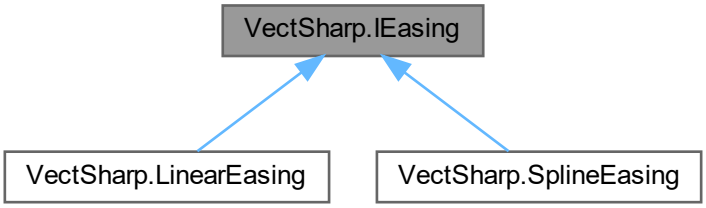
The documentation for this interface was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.71 VectSharp.IEasing Interface Reference

Describes a function used to transform the transition speed.

Inheritance diagram for VectSharp.IEasing:



Public Member Functions

- double Ease (double value)

  Applies the easing to the specified transition offset.

7.71.1 Detailed Description

Describes a function used to transform the transition speed.

Definition at line 1220 of file Animation.cs.

7.71.2 Member Function Documentation

7.71.2.1 Ease()

```
double VectSharp.IEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implemented in VectSharp.SplineEasing, and VectSharp.LinearEasing.

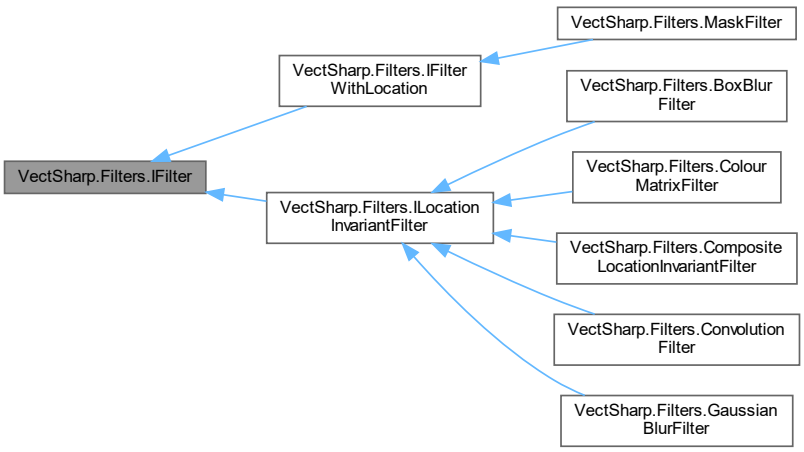
The documentation for this interface was generated from the following file:

- VectSharp/Animation.cs

7.72 VectSharp.Filters.IFilter Interface Reference

Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.

Inheritance diagram for VectSharp.Filters.IFilter:



Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.72.1 Detailed Description

Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.

Definition at line 25 of file Filters.cs.

7.72.2 Property Documentation

7.72.2.1 BottomRightMargin

```
Point VectSharp.Filters.IFilter.BottomRightMargin [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil VectSharp.Filters.ConvolutionFilter, VectSharp.Filters.GaussianBlurFilter, and VectSharp.Filters.MaskFilter.

Definition at line 35 of file Filters.cs.

7.72.2.2 TopLeftMargin

```
Point VectSharp.Filters.IFilter.TopLeftMargin [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil VectSharp.Filters.ConvolutionFilter, VectSharp.Filters.GaussianBlurFilter, and VectSharp.Filters.MaskFilter.

Definition at line 30 of file Filters.cs.

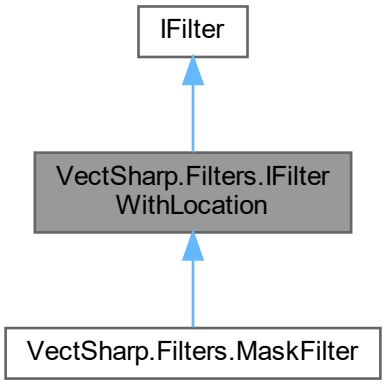
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.73 VectSharp.Filters.IFilterWithLocation Interface Reference

Represents a filter whose results depend on the position of the subject image on the graphics surface.

Inheritance diagram for VectSharp.Filters.IFilterWithLocation:



Public Member Functions

- RasterImage Filter (RasterImage image, Rectangle bounds, double scale)
  Applies the filter to a RasterImage.

Additional Inherited Members

7.73.1 Detailed Description

Represents a filter whose results depend on the position of the subject image on the graphics surface.

Definition at line 55 of file Filters.cs.

7.73.2 Member Function Documentation

7.73.2.1 Filter()

```
RasterImage VectSharp.Filters.IFilterWithLocation.Filter (
            RasterImage image,
            Rectangle bounds,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
bounds	The region on the graphics surface where the image will be drawn.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implemented in VectSharp.Filters.MaskFilter.

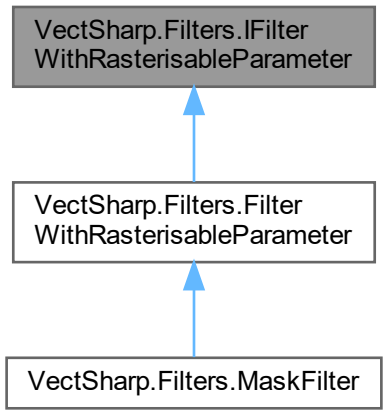
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.74 VectSharp.Filters.IFilterWithRasterisableParameter Interface Reference

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.

Inheritance diagram for VectSharp.Filters.IFilterWithRasterisableParameter:



Public Member Functions

- void RasteriseParameter (Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod, double scale)

  Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

7.74.1 Detailed Description

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.

Definition at line 71 of file Filters.cs.

7.74.2 Member Function Documentation

7.74.2.1 RasteriseParameter()

```
void VectSharp.Filters.IFilterWithRasterisableParameter.RasteriseParameter (
            Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod,
            double scale )
```

Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.
scale	The scale factor at which the parameter is rasterised.

Implemented in VectSharp.Filters.FilterWithRasterisableParameter.

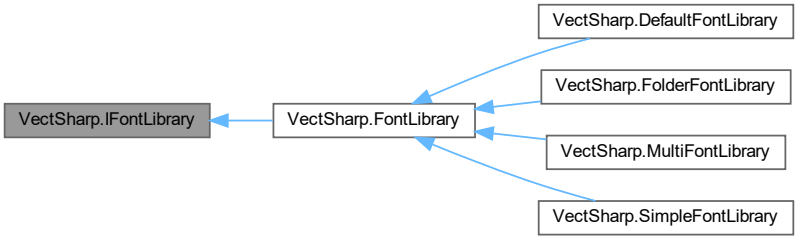
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.75 VectSharp.IFontLibrary Interface Reference

Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.

Inheritance diagram for VectSharp.IFontLibrary:



Public Member Functions

- FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.
- FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised.
- FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.
- FontFamily ResolveFontFamily (string fontFamily, FontFamily.StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback .

7.75.1 Detailed Description

Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.

Definition at line 28 of file FontLibrary.cs.

7.75.2 Member Function Documentation

7.75.2.1 ResolveFontFamily() [1/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implemented in VectSharp.FontLibrary, VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.75.2.2 ResolveFontFamily() [2/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implemented in VectSharp.FontLibrary, VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.75.2.3 ResolveFontFamily() [3/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily,
            FontFamily.StandardFontFamilies finalFallback,
            params string[] fallback )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* .

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implemented in VectSharp.FontLibrary.

7.75.2.4 ResolveFontFamily() [4/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implemented in VectSharp.FontLibrary.

The documentation for this interface was generated from the following file:

- VectSharp/FontLibrary.cs

7.76 VectSharp.IGraphicsContext Interface Reference

This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.

Public Member Functions

- void **Save** ()

  Save the current transform state (rotation, translation, scale). This should be implemented as a LIFO stack.
- void **Restore** ()

  Restore the previous transform state (rotation, translation, scale). This should be implemented as a LIFO stack.
- void Translate (double x, double y)

  Translate the coordinate system origin.
- void Rotate (double angle)

  Rotate the coordinate system around the origin.
- void Scale (double scaleX, double scaleY)

  Scale the coordinate system with respect to the origin.
- void Transform (double a, double b, double c, double d, double e, double f)

  Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].
- void FillText (string text, double x, double y)

  Fill a text string using the current Font and TextBaseline.
- void StrokeText (string text, double x, double y)

  Stroke the outline of a text string using the current Font and TextBaseline.
- void MoveTo (double x, double y)

  Change the current point without drawing a line from the previous point. If necessary, start a new figure.
- void LineTo (double x, double y)

  Draw a line from the previous point to the specified point.
- void **Close** ()

  Close the current figure.
- void **Stroke** ()

Stroke the current path using the current StrokeStyle, LineWidth, LineCap, LineJoin and LineDash.

- void **SetClippingPath** ()

  Set the current clipping path as the intersection of the previous clipping path and the current path.
- void SetFillStyle ((int r, int g, int b, double a) style)

  Set the current FillStyle.
- void SetFillStyle (Brush style)

  Set the current FillStyle.
- void SetStrokeStyle ((int r, int g, int b, double a) style)

  Set the current StrokeStyle.
- void SetStrokeStyle (Brush style)

  Set the current StrokeStyle.
- void CubicBezierTo (double p1X, double p1Y, double p2X, double p2Y, double p3X, double p3Y)

  Add to the current figure a cubic Bezier from the current point to a destination point, with two control points.
- void Rectangle (double x0, double y0, double width, double height)

  Add a rectangle figure to the current path.
- void Fill (FillRule fillRule)

  Fill the current path using the current FillStyle.
- void SetLineDash (LineDash dash)

  Set the current line dash pattern.
- void DrawRasterImage (int sourceX, int sourceY, int sourceWidth, int sourceHeight, double destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage image)

  Draw a raster image.
- void DrawFilteredGraphics (Graphics graphics, IFilter filter)

  Draws a Graphics object, applying the specified filter .

Properties

- double Width `[get]`

  Width of the graphic surface.
- double Height `[get]`

  Height of the graphic surface.
- Font Font `[get, set]`

  The current font.
- TextBaselines TextBaseline `[get, set]`

  The current text baseline.
- Brush FillStyle `[get]`

  Current brush used to fill paths.
- Brush StrokeStyle `[get]`

  Current brush used to stroke paths.
- double LineWidth `[get, set]`

  Current line width used to stroke paths.
- LineCaps LineCap `[set]`

  Current line cap used to stroke paths.
- LineJoins LineJoin `[set]`

  Current line join used to stroke paths.
- string Tag `[get, set]`

  The current tag. How this can be used depends on each implementation.

7.76.1 Detailed Description

This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.

Definition at line 35 of file Graphics.cs.

7.76.2 Member Function Documentation

7.76.2.1 CubicBezierTo()

```
void VectSharp.IGraphicsContext.CubicBezierTo (
            double p1X,
            double p1Y,
            double p2X,
            double p2Y,
            double p3X,
            double p3Y )
```

Add to the current figure a cubic Bezier from the current point to a destination point, with two control points.

Parameters

p1X	The horizontal coordinate of the first control point.
p1Y	The vertical coordinate of the first control point.
p2X	The horizontal coordinate of the second control point.
p2Y	The vertical coordinate of the second control point.
p3X	The horizontal coordinate of the destination point.
p3Y	The vertical coordinate of the destination point.

7.76.2.2 DrawFilteredGraphics()

```
void VectSharp.IGraphicsContext.DrawFilteredGraphics (
            Graphics graphics,
            IFilter filter )
```

Draws a Graphics object, applying the specified *filter* .

Parameters

graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.

7.76.2.3 DrawRasterImage()

```
void VectSharp.IGraphicsContext.DrawRasterImage (
            int sourceX,
            int sourceY,
            int sourceWidth,
            int sourceHeight,
            double destinationX,
            double destinationY,
            double destinationWidth,
            double destinationHeight,
            RasterImage image )
```

Draw a raster image.

Parameters

sourceX	The horizontal coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceY	The vertical coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceWidth	The width of the rectangle delimiting the source area of the image.
sourceHeight	The height of the rectangle delimiting the source area of the image.
destinationX	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationY	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationWidth	The width of the rectangle delimiting the destination area of the image.
destinationHeight	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.

7.76.2.4 Fill()

```
void VectSharp.IGraphicsContext.Fill (
            FillRule fillRule )
```

Fill the current path using the current FillStyle.

Parameters

fillRule	The FillRule that determines which parts of the path are filled.

7.76.2.5 FillText()

```
void VectSharp.IGraphicsContext.FillText (
            string text,
```

```
            double x,
            double y )
```

Fill a text string using the current Font and TextBaseline.

Parameters

text	The string to draw.
x	The horizontal coordinate of the text origin.
y	The vertical coordinate of the text origin.

7.76.2.6 LineTo()

```
void VectSharp.IGraphicsContext.LineTo (
            double x,
            double y )
```

Draw a line from the previous point to the specified point.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

7.76.2.7 MoveTo()

```
void VectSharp.IGraphicsContext.MoveTo (
            double x,
            double y )
```

Change the current point without drawing a line from the previous point. If necessary, start a new figure.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

7.76.2.8 Rectangle()

```
void VectSharp.IGraphicsContext.Rectangle (
            double x0,
```

```
            double y0,
            double width,
            double height )
```

Add a rectangle figure to the current path.

Parameters

x0	The horizontal coordinate of the top-left corner of the rectangle.
y0	The vertical coordinate of the top-left corner of the rectangle.
width	The width of corner of the rectangle.
height	The height of corner of the rectangle.

7.76.2.9 Rotate()

```
void VectSharp.IGraphicsContext.Rotate (
            double angle )
```

Rotate the coordinate system around the origin.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.

7.76.2.10 Scale()

```
void VectSharp.IGraphicsContext.Scale (
            double scaleX,
            double scaleY )
```

Scale the coordinate system with respect to the origin.

Parameters

scaleX	The horizontal scale.
scaleY	The vertical scale.

7.76.2.11 SetFillStyle() [1/2]

```
void VectSharp.IGraphicsContext.SetFillStyle (
            (int r, int g, int b, double a) style )
```

Set the current FillStyle.

Parameters

style	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

7.76.2.12 SetFillStyle() [2/2]

```
void VectSharp.IGraphicsContext.SetFillStyle (
            Brush style )
```

Set the current FillStyle.

Parameters

style	The new fill style.

7.76.2.13 SetLineDash()

```
void VectSharp.IGraphicsContext.SetLineDash (
            LineDash dash )
```

Set the current line dash pattern.

Parameters

dash	The line dash pattern.

7.76.2.14 SetStrokeStyle() [1/2]

```
void VectSharp.IGraphicsContext.SetStrokeStyle (
            (int r, int g, int b, double a) style )
```

Set the current StrokeStyle.

Parameters

style	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

7.76.2.15 SetStrokeStyle() [2/2]

```
void VectSharp.IGraphicsContext.SetStrokeStyle (
            Brush style )
```

Set the current StrokeStyle.

Parameters

style	The new stroke style.

7.76.2.16 StrokeText()

```
void VectSharp.IGraphicsContext.StrokeText (
            string text,
            double x,
            double y )
```

Stroke the outline of a text string using the current Font and TextBaseline.

Parameters

text	The string to draw.
x	The horizontal coordinate of the text origin.
y	The vertical coordinate of the text origin.

7.76.2.17 Transform()

```
void VectSharp.IGraphicsContext.Transform (
            double a,
            double b,
            double c,
            double d,
            double e,
            double f )
```

Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].

Parameters

a	The first element of the first column.
b	The second element of the first column.
c	The first element of the second column.
d	The second element of the second column.
e	The first element of the third column.
f	The second element of the third column.

7.76.2.18 Translate()

```
void VectSharp.IGraphicsContext.Translate (
            double x,
            double y )
```

Translate the coordinate system origin.

Parameters

x	The horizontal translation.
y	The vertical translation.

7.76.3 Property Documentation

7.76.3.1 FillStyle

```
Brush VectSharp.IGraphicsContext.FillStyle  [get]
```

Current brush used to fill paths.

Definition at line 146 of file Graphics.cs.

7.76.3.2 Font

```
Font VectSharp.IGraphicsContext.Font  [get], [set]
```

The current font.

Definition at line 91 of file Graphics.cs.

7.76.3.3 Height

```
double VectSharp.IGraphicsContext.Height  [get]
```

Height of the graphic surface.

Definition at line 45 of file Graphics.cs.

7.76.3.4 LineCap

```
LineCaps VectSharp.IGraphicsContext.LineCap  [set]
```

Current line cap used to stroke paths.

Definition at line 211 of file Graphics.cs.

7.76.3.5 LineJoin

```
LineJoins VectSharp.IGraphicsContext.LineJoin  [set]
```

Current line join used to stroke paths.

Definition at line 216 of file Graphics.cs.

7.76.3.6 LineWidth

```
double VectSharp.IGraphicsContext.LineWidth  [get], [set]
```

Current line width used to stroke paths.

Definition at line 206 of file Graphics.cs.

7.76.3.7 StrokeStyle

```
Brush VectSharp.IGraphicsContext.StrokeStyle  [get]
```

Current brush used to stroke paths.

Definition at line 163 of file Graphics.cs.

7.76.3.8 Tag

```
string VectSharp.IGraphicsContext.Tag  [get], [set]
```

The current tag. How this can be used depends on each implementation.

Definition at line 227 of file Graphics.cs.

7.76.3.9 TextBaseline

```
TextBaselines VectSharp.IGraphicsContext.TextBaseline  [get], [set]
```

The current text baseline.

Definition at line 96 of file Graphics.cs.

7.76.3.10 Width

```
double VectSharp.IGraphicsContext.Width  [get]
```

Width of the graphic surface.

Definition at line 40 of file Graphics.cs.

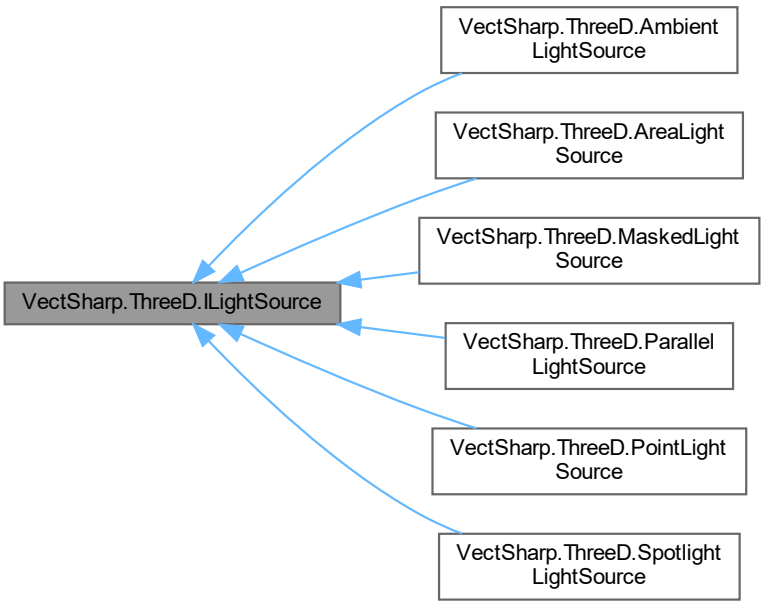
The documentation for this interface was generated from the following file:

- VectSharp/Graphics.cs

7.77 VectSharp.ThreeD.ILightSource Interface Reference

Represents a light source.

Inheritance diagram for VectSharp.ThreeD.ILightSource:



Public Member Functions

- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.
- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Properties

- bool CastsShadow `[get]`

  Determines whether the light casts a shadow or not.

7.77.1 Detailed Description

Represents a light source.

Definition at line 65 of file Lights.cs.

7.77.2 Member Function Documentation

7.77.2.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.ILightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

7.77.2.2 GetObstruction()

```
double VectSharp.ThreeD.ILightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

7.77.3 Property Documentation

7.77.3.1 CastsShadow

```
bool VectSharp.ThreeD.ILightSource.CastsShadow  [get]
```

Determines whether the light casts a shadow or not.

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

Definition at line 77 of file Lights.cs.

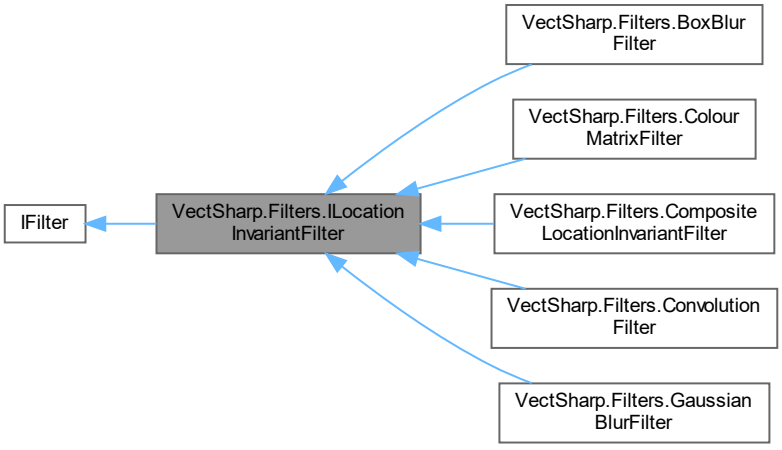
The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.78 VectSharp.Filters.ILocationInvariantFilter Interface Reference

Represents a filter that can be applied to an image regardless of its location on the graphics surface.

Inheritance diagram for VectSharp.Filters.ILocationInvariantFilter:



Public Member Functions

- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Additional Inherited Members

7.78.1 Detailed Description

Represents a filter that can be applied to an image regardless of its location on the graphics surface.

Definition at line 41 of file Filters.cs.

7.78.2 Member Function Documentation

7.78.2.1 Filter()

```
RasterImage VectSharp.Filters.ILocationInvariantFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil
VectSharp.Filters.ConvolutionFilter, and VectSharp.Filters.GaussianBlurFilter.

The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.79 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter Class Reference

Contains methods to render a Page to an Image.

Classes

- class UnknownFormatException

  The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Static Public Member Functions

- static Image< SixLabors.ImageSharp.PixelFormats.Rgba32 > SaveAsImage (this Page page, double scale=1)

  Render the page to an Image object.
- static void SaveAsImage (this Page page, Stream imageStream, OutputFormats outputFormat, double scale=1)

  Render the page to an image stream.
- static void SaveAsImage (this Page page, string fileName, OutputFormats? outputFormat=null, double scale=1)

  Render the page to an image file.
- static DisposableIntPtr SaveAsRawBytes (this Page pag, out int width, out int height, out int totalSize, double scale=1)

  Render the page to raw pixel data, in 32bpp RGBA format.
- static byte[ ] SaveAsRawBytes (this Page pag, out int width, out int height, double scale=1)

  Return the page to raw pixel data, in 32bpp RGBA format.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.
- static Image SaveAsAnimatedGIF (this Animation animation, double scale=1, double frameRate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF.
- static void SaveAsAnimatedGIF (this Animation animation, Stream imageStream, double scale=1, double frameRate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF stream.
- static void SaveAsAnimatedGIF (this Animation animation, string fileName, double scale=1, double frameRate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF file.
- static void SaveAsAnimatedPNG (this Animation animation, Stream imageStream, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframeCompression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to a stream in animated PNG format.
- static void SaveAsAnimatedPNG (this Animation animation, string fileName, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframeCompression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to an animated PNG file.

7.79.1 Detailed Description

Contains methods to render a Page to an Image.

Definition at line 1051 of file ImageSharpContext.cs.

7.79.2 Member Function Documentation

7.79.2.1 Rasterise()

```
static RasterImage VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 1283 of file ImageSharpContext.cs.

7.79.2.2 SaveAsAnimatedGIF() [1/3]

```
static Image VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF.

Parameters

animation	The animation to export.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Returns

An Image containing the animated GIF.

Definition at line 1331 of file ImageSharpContext.cs.

7.79.2.3 SaveAsAnimatedGIF() [2/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF stream.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated GIF will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Definition at line 1390 of file ImageSharpContext.cs.

7.79.2.4 SaveAsAnimatedGIF() [3/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            string fileName,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF file.

Parameters

animation	The animation to export.
fileName	The output file to create.

Parameters

scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Definition at line 1406 of file ImageSharpContext.cs.

7.79.2.5 SaveAsAnimatedPNG() [1/2]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedPNG (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to a stream in animated PNG format.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated PNG will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 1423 of file ImageSharpContext.cs.

7.79.2.6 SaveAsAnimatedPNG() [2/2]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedPNG (
            this Animation animation,
            string fileName,
```

```
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to an animated PNG file.

Parameters

animation	The animation to export.
fileName	The output file to create.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 1511 of file ImageSharpContext.cs.

7.79.2.7 SaveAsImage() [1/3]

```
static Image< SixLabors.ImageSharp.PixelFormats.Rgba32 > VectSharp.Raster.ImageSharp.Image↩
SharpContextInterpreter.SaveAsImage (
            this Page page,
            double scale = 1 )  [static]
```

Render the page to an Image object.

Parameters

page	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the Image.

Returns

An Image containing the rasterised page.

Definition at line 1059 of file ImageSharpContext.cs.

7.79.2.8 SaveAsImage() [2/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsImage (
            this Page page,
```

```
            Stream imageStream,
            OutputFormats outputFormat,
            double scale = 1 )  [static]
```

Render the page to an image stream.

Parameters

page	The Page to render.
imageStream	The Stream on which the image data will be written.
outputFormat	The format of the image that will be created.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.

Definition at line 1075 of file ImageSharpContext.cs.

7.79.2.9 SaveAsImage() [3/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsImage (
            this Page page,
            string fileName,
            OutputFormats?  outputFormat = null,
            double scale = 1 )  [static]
```

Render the page to an image file.

Parameters

page	The Page to render.
fileName	The path of the file where the image will be saved.
outputFormat	The format of the image that will be created. If this is `null` (the default), the format is desumed from the extension of the file.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.

Definition at line 1133 of file ImageSharpContext.cs.

7.79.2.10 SaveAsRawBytes() [1/2]

```
static byte[] VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsRawBytes (
            this Page pag,
            out int width,
            out int height,
            double scale = 1 )  [static]
```

Return the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.

Returns

A byte array containing the raw pixel data.

Definition at line 1246 of file ImageSharpContext.cs.

7.79.2.11 SaveAsRawBytes() [2/2]

```
static DisposableIntPtr VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsRaw↩
Bytes (
            this Page pag,
            out int width,
            out int height,
            out int totalSize,
            double scale = 1 )  [static]
```

Render the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.
totalSize	The size in bytes of the raw pixel data.

Returns

A DisposableIntPtr containing a pointer to the raw pixel data, stored in unmanaged memory. Dispose this object to release the unmanaged memory.

Definition at line 1203 of file ImageSharpContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Raster.ImageSharp/ImageSharpContext.cs

7.80 VectSharp.ImageSharpUtils.ImageURIParser Class Reference

Provides a method to parse an image URI into a page.

Static Public Member Functions

- static Func< string, bool, Page > Parser (Func< string, bool, Page > parseSVG)

  Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

7.80.1 Detailed Description

Provides a method to parse an image URI into a page.

Definition at line 29 of file ImageUriParser.cs.

7.80.2 Member Function Documentation

7.80.2.1 Parser()

```
static Func< string, bool, Page > VectSharp.ImageSharpUtils.ImageURIParser.Parser (
            Func< string, bool, Page > parseSVG )  [static]
```

Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.ImageSharpUtils.ImageURIParser.Parser(Vec
```

Parameters

parseSVG	A function to parse an SVG image uri into a page. You should pass `VectSharp.SVG.Parser.ParseSVGURI` as this argument.

Returns

A function to parse an image URI into a page.

Definition at line 37 of file ImageUriParser.cs.

The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/ImageUriParser.cs

7.81 VectSharp.MuPDFUtils.ImageURIParser Class Reference

Provides a method to parse an image URI into a page.

Static Public Member Functions

- static Func< string, bool, Page > Parser (Func< string, bool, Page > parseSVG)

  Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

7.81.1 Detailed Description

Provides a method to parse an image URI into a page.

Definition at line 29 of file ImageURIParser.cs.

7.81.2 Member Function Documentation

7.81.2.1 Parser()

```
static Func< string, bool, Page > VectSharp.MuPDFUtils.ImageURIParser.Parser (
            Func< string, bool, Page > parseSVG )  [static]
```

Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.MuPDFUtils.ImageURIParser.Parser(VectShar
```

Parameters

parseSVG	A function to parse an SVG image uri into a page. You should pass `VectSharp.SVG.Parser.ParseSVGURI` as this argument.

Returns

A function to parse an image URI into a page.

Definition at line 37 of file ImageURIParser.cs.

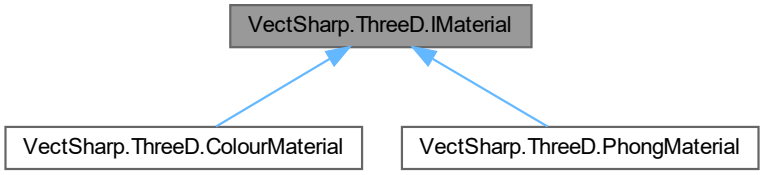
The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/ImageURIParser.cs

7.82 VectSharp.ThreeD.IMaterial Interface Reference

Represents a material used to the determine the appearance of Triangle3DElement.

Inheritance diagram for VectSharp.ThreeD.IMaterial:



Public Member Functions

- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

7.82.1 Detailed Description

Represents a material used to the determine the appearance of Triangle3DElement.

Definition at line 31 of file Materials.cs.

7.82.2 Member Function Documentation

7.82.2.1 GetColour()

```
Colour VectSharp.ThreeD.IMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implemented in VectSharp.ThreeD.ColourMaterial, and VectSharp.ThreeD.PhongMaterial.

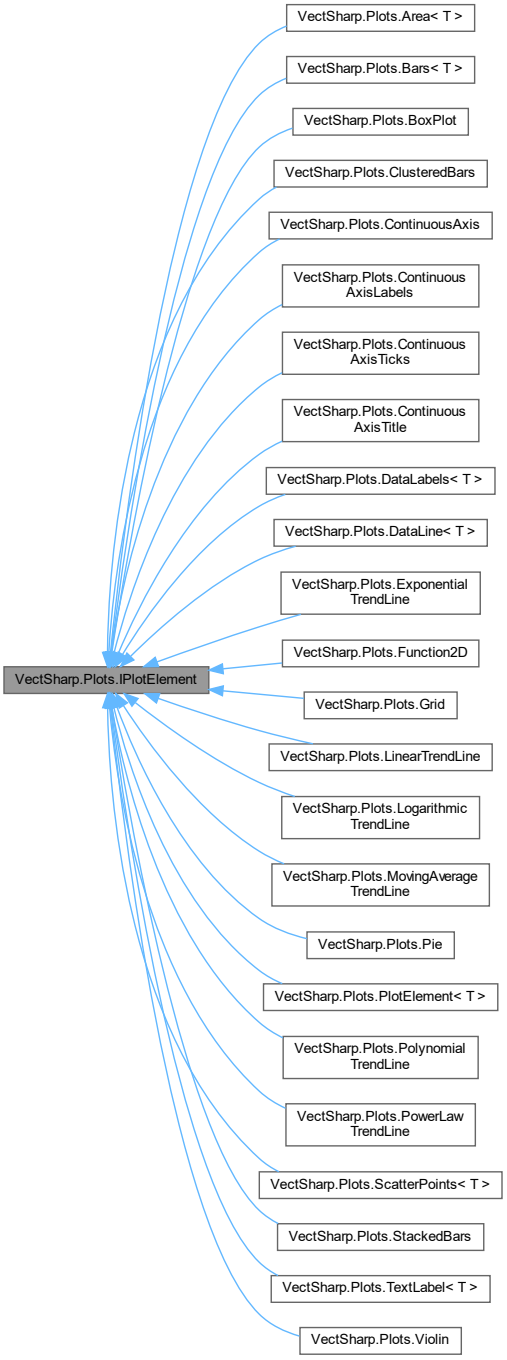
The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.83 VectSharp.Plots.IPlotElement Interface Reference

Represents a plot element.

Inheritance diagram for VectSharp.Plots.IPlotElement:



Public Member Functions

- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Properties

- ICoordinateSystem CoordinateSystem [get]

  The coordinate system used to transform the points from data space to plot space.

7.83.1 Detailed Description

Represents a plot element.

Definition at line 91 of file Plot.cs.

7.83.2 Member Function Documentation

7.83.2.1 Plot()

```
void VectSharp.Plots.IPlotElement.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implemented in VectSharp.Plots.ContinuousAxis, VectSharp.Plots.ContinuousAxisTicks, VectSharp.Plots.ContinuousAxisLabels, VectSharp.Plots.ContinuousAxisTitle, VectSharp.Plots.Grid, VectSharp.Plots.Bars< T >, VectSharp.Plots.StackedBars, VectSharp.Plots.ClusteredBars, VectSharp.Plots.BoxPlot, VectSharp.Plots.ScatterPoints< T >, VectSharp.Plots.TextLabel< T >, VectSharp.Plots.DataLabels< T >, VectSharp.Plots.DataLine< T >, VectSharp.Plots.Area< T >, VectSharp.Plots.Function2D, VectSharp.Plots.Pie, VectSharp.Plots.PlotElement< T >, VectSharp.Plots.LinearTrendLine, VectSharp.Plots.ExponentialTrendLine, VectSharp.Plots.LogarithmicTrendLine, VectSharp.Plots.PolynomialTrendLine, VectSharp.Plots.PowerLawTrendLine, VectSharp.Plots.MovingAverageTrendLine, and VectSharp.Plots.Violin.

7.83.3 Property Documentation

7.83.3.1 CoordinateSystem

```
ICoordinateSystem VectSharp.Plots.IPlotElement.CoordinateSystem [get]
```

The coordinate system used to transform the points from data space to plot space.

Implemented in VectSharp.Plots.ContinuousAxis, VectSharp.Plots.ContinuousAxisTicks, VectSharp.Plots.ContinuousAxisLabels, VectSharp.Plots.ContinuousAxisTitle, VectSharp.Plots.Grid, VectSharp.Plots.Bars< T >, VectSharp.Plots.StackedBars, VectSharp.Plots.ClusteredBars, VectSharp.Plots.BoxPlot, VectSharp.Plots.ScatterPoints< T >, VectSharp.Plots.TextLabel< T >,

VectSharp.Plots.DataLabels< T >, VectSharp.Plots.DataLine< T >, VectSharp.Plots.Area< T >, VectSharp.Plots.Function2D, VectSharp.Plots.Pie, VectSharp.Plots.PlotElement< T >, VectSharp.Plots.LinearTrendLine, VectSharp.Plots.ExponentialTrendLine, VectSharp.Plots.LogarithmicTrendLine, VectSharp.Plots.PolynomialTrendLine, VectSharp.Plots.PowerLawTrendLine, VectSharp.Plots.MovingAverageTrendLine, and VectSharp.Plots.Violin.

Definition at line 102 of file Plot.cs.

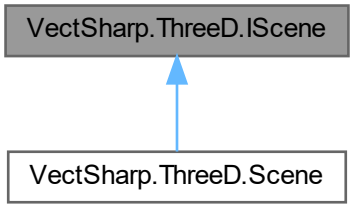
The documentation for this interface was generated from the following file:

- VectSharp.Plots/Plot.cs

7.84 VectSharp.ThreeD.IScene Interface Reference

Represents a 3D scene.

Inheritance diagram for VectSharp.ThreeD.IScene:



Public Member Functions

- void AddElement (Element3D element)

  Adds the specified element to the scene.
- void AddRange (IEnumerable< Element3D > elements)

  Adds the specified elements to the scene.
- void Replace (Func< Element3D, Element3D > replacementFunction)

  Replaces each element in the scene with the element returned by the replacementFunction .
- void Replace (Func< Element3D, IEnumerable< Element3D > > replacementFunction)

  Replaces each element in the scene with the element(s) returned by the replacementFunction .

Properties

- IEnumerable< Element3D > SceneElements `[get]`

  The Element3Ds constituting the scene.
- object SceneLock `[get]`

  An object used to synchronise multithreaded rendering of the same scene.

7.84.1 Detailed Description

Represents a 3D scene.

Definition at line 26 of file Scene.cs.

7.84.2 Member Function Documentation

7.84.2.1 AddElement()

```
void VectSharp.ThreeD.IScene.AddElement (
            Element3D element )
```

Adds the specified *element* to the scene.

Parameters

element	The Element3D to add.

Implemented in VectSharp.ThreeD.Scene.

7.84.2.2 AddRange()

```
void VectSharp.ThreeD.IScene.AddRange (
            IEnumerable< Element3D > elements )
```

Adds the specified *elements* to the scene.

Parameters

elements	A collection of Element3Ds to add.

Implemented in VectSharp.ThreeD.Scene.

7.84.2.3 Replace() [1/2]

```
void VectSharp.ThreeD.IScene.Replace (
            Func< Element3D, Element3D > replacementFunction )
```

Replaces each element in the scene with the element returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with another Element3D.

Implemented in VectSharp.ThreeD.Scene.

7.84.2.4 Replace() [2/2]

```
void VectSharp.ThreeD.IScene.Replace (
            Func< Element3D, IEnumerable< Element3D > > replacementFunction )
```

Replaces each element in the scene with the element(s) returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with 0 or more Element3Ds.

Implemented in VectSharp.ThreeD.Scene.

7.84.3 Property Documentation

7.84.3.1 SceneElements

```
IEnumerable<Element3D> VectSharp.ThreeD.IScene.SceneElements  [get]
```

The Element3Ds constituting the scene.

Implemented in VectSharp.ThreeD.Scene.

Definition at line 31 of file Scene.cs.

7.84.3.2 SceneLock

```
object VectSharp.ThreeD.IScene.SceneLock  [get]
```

An object used to synchronise multithreaded rendering of the same scene.

Implemented in VectSharp.ThreeD.Scene.

Definition at line 60 of file Scene.cs.

The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Scene.cs

7.85 VectSharp.ThreeD.LightIntensity Struct Reference

Represents the intensity of a light source at a particular point.

Public Member Functions

- LightIntensity (double intensity, NormalizedVector3D direction)

  Creates a new LightIntensity.
- void Deconstruct (out double intensity, out NormalizedVector3D direction)

  Deconstructs the struct.

Public Attributes

- double Intensity

  The intensity of the light.
- NormalizedVector3D Direction

  The direction towards from which the light comes.

7.85.1 Detailed Description

Represents the intensity of a light source at a particular point.

Definition at line 27 of file Lights.cs.

7.85.2 Constructor & Destructor Documentation

7.85.2.1 LightIntensity()

```
VectSharp.ThreeD.LightIntensity.LightIntensity (
            double intensity,
            NormalizedVector3D direction )
```

Creates a new LightIntensity.

Parameters

intensity	The intensity of the light.
direction	The direction from which the light comes.

Definition at line 44 of file Lights.cs.

7.85.3 Member Function Documentation

7.85.3.1 Deconstruct()

```
void VectSharp.ThreeD.LightIntensity.Deconstruct (
            out double intensity,
            out NormalizedVector3D direction )
```

Deconstructs the struct.

Parameters

intensity	This parameter will hold the Intensity of the light.
direction	This parameter will hold the Direction of the light.

Definition at line 55 of file Lights.cs.

7.85.4 Member Data Documentation

7.85.4.1 Direction

```
NormalizedVector3D VectSharp.ThreeD.LightIntensity.Direction
```

The direction towards from which the light comes.

Definition at line 37 of file Lights.cs.

7.85.4.2 Intensity

```
double VectSharp.ThreeD.LightIntensity.Intensity
```

The intensity of the light.

Definition at line 32 of file Lights.cs.

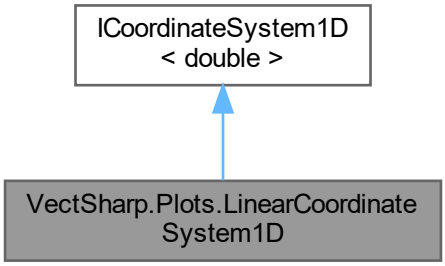
The documentation for this struct was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.86 VectSharp.Plots.LinearCoordinateSystem1D Class Reference

Represents a 1-D linear coordinate system.

Inheritance diagram for VectSharp.Plots.LinearCoordinateSystem1D:



Public Member Functions

- LinearCoordinateSystem1D (double min, double max, double scale)

  Creates a new LinearCoordinateSystem1D, manually specifying the parameters.
- LinearCoordinateSystem1D (IReadOnlyList< double > data, double scale=350)

  Creates a new LinearCoordinateSystem1D determining the value range from the data .
- double ToPlotCoordinates (double dataPoint)

Properties

- double Min `[get, set]`

  The minimum value.
- double Max `[get, set]`

  The maximum value.
- double Scale `[get, set]`

  The scale factor.

7.86.1 Detailed Description

Represents a 1-D linear coordinate system.

Definition at line 923 of file CoordinateSystems.cs.

7.86.2 Constructor & Destructor Documentation

7.86.2.1 LinearCoordinateSystem1D() [1/2]

```
VectSharp.Plots.LinearCoordinateSystem1D.LinearCoordinateSystem1D (
            double min,
            double max,
            double scale )
```

Creates a new LinearCoordinateSystem1D, manually specifying the parameters.

Parameters

min	The minimum value.
max	The maximum value.
scale	The scale factor.

Definition at line 946 of file CoordinateSystems.cs.

7.86.2.2 LinearCoordinateSystem1D() [2/2]

```
VectSharp.Plots.LinearCoordinateSystem1D.LinearCoordinateSystem1D (
            IReadOnlyList< double > data,
            double scale = 350 )
```

Creates a new LinearCoordinateSystem1D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scale	The scale factor.

Definition at line 958 of file CoordinateSystems.cs.

7.86.3 Member Function Documentation

7.86.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.LinearCoordinateSystem1D.ToPlotCoordinates (
            double dataPoint )
```

Definition at line 978 of file CoordinateSystems.cs.

7.86.4 Property Documentation

7.86.4.1 Max

```
double VectSharp.Plots.LinearCoordinateSystem1D.Max  [get], [set]
```

The maximum value.

Definition at line 933 of file CoordinateSystems.cs.

7.86.4.2 Min

```
double VectSharp.Plots.LinearCoordinateSystem1D.Min  [get], [set]
```

The minimum value.

Definition at line 928 of file CoordinateSystems.cs.

7.86.4.3 Scale

```
double VectSharp.Plots.LinearCoordinateSystem1D.Scale  [get], [set]
```

The scale factor.

Definition at line 938 of file CoordinateSystems.cs.

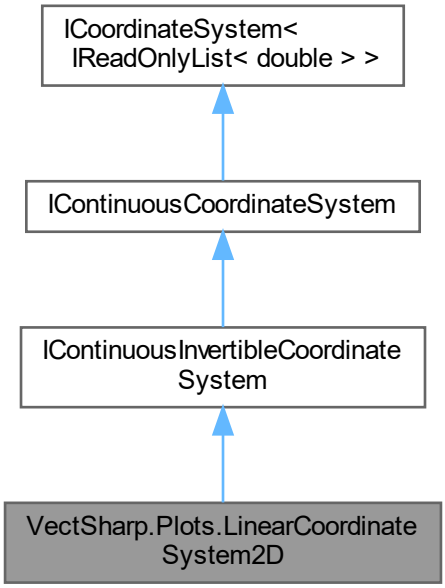
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.87 VectSharp.Plots.LinearCoordinateSystem2D Class Reference

Represents a linear coordinate system.

Inheritance diagram for VectSharp.Plots.LinearCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LinearCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LinearCoordinateSystem2D manually specifying the parameter values.

- LinearCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LinearCoordinateSystem2D determining the value range from the data .

- LinearCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LinearCoordinateSystem2D determining the value range from the data .

- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value.
- double MaxY `[get, set]`

  The maximum Y value.
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.87.1 Detailed Description

Represents a linear coordinate system.

Definition at line 133 of file CoordinateSystems.cs.

7.87.2 Constructor & Destructor Documentation

7.87.2.1 LinearCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LinearCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 196 of file CoordinateSystems.cs.

7.87.2.2 LinearCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinearCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 212 of file CoordinateSystems.cs.

7.87.2.3 LinearCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinearCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 248 of file CoordinateSystems.cs.

7.87.3 Member Function Documentation

7.87.3.1 GetAround()

```
double[] VectSharp.Plots.LinearCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 301 of file CoordinateSystems.cs.

7.87.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LinearCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

7.87.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LinearCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 279 of file CoordinateSystems.cs.

7.87.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LinearCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 285 of file CoordinateSystems.cs.

7.87.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LinearCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 295 of file CoordinateSystems.cs.

7.87.4 Property Documentation

7.87.4.1 IsLinear

```
bool VectSharp.Plots.LinearCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 166 of file CoordinateSystems.cs.

7.87.4.2 MaxX

```
double VectSharp.Plots.LinearCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 143 of file CoordinateSystems.cs.

7.87.4.3 MaxY

```
double VectSharp.Plots.LinearCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value.

Definition at line 153 of file CoordinateSystems.cs.

7.87.4.4 MinX

```
double VectSharp.Plots.LinearCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value.

Definition at line 138 of file CoordinateSystems.cs.

7.87.4.5 MinY

```
double VectSharp.Plots.LinearCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value.

Definition at line 148 of file CoordinateSystems.cs.

7.87.4.6 Resolution

```
double [] VectSharp.Plots.LinearCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 174 of file CoordinateSystems.cs.

7.87.4.7 ScaleX

```
double VectSharp.Plots.LinearCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 158 of file CoordinateSystems.cs.

7.87.4.8 ScaleY

```
double VectSharp.Plots.LinearCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 163 of file CoordinateSystems.cs.

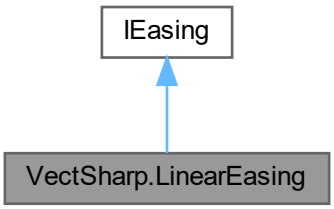
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.88 VectSharp.LinearEasing Class Reference

Describes a linear easing (i.e., no easing).

Inheritance diagram for VectSharp.LinearEasing:



Public Member Functions

- LinearEasing ()

  Creates a new LinearEasing.
- double Ease (double value)

  Applies the easing to the specified transition offset.

Parameters

value	*The transition offset (ranging from 0 to 1).*

Returns

The eased transition offset value.

7.88.1 Detailed Description

Describes a linear easing (i.e., no easing).

Definition at line 1324 of file Animation.cs.

7.88.2 Constructor & Destructor Documentation

7.88.2.1 LinearEasing()

```
VectSharp.LinearEasing.LinearEasing ( )
```

Creates a new LinearEasing.

Definition at line 1329 of file Animation.cs.

7.88.3 Member Function Documentation

7.88.3.1 Ease()

```
double VectSharp.LinearEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implements VectSharp.IEasing.

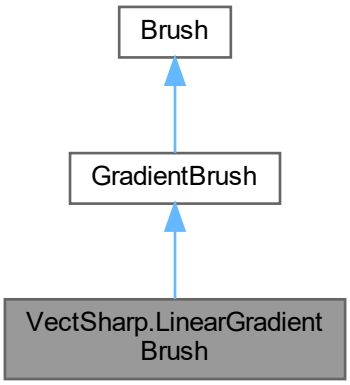
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.89 VectSharp.LinearGradientBrush Class Reference

Represents a brush painting with a linear gradient.

Inheritance diagram for VectSharp.LinearGradientBrush:



Public Member Functions

- LinearGradientBrush (Point startPoint, Point endPoint, IEnumerable< GradientStop > gradientStops)

  Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.
- LinearGradientBrush (Point startPoint, Point endPoint, params GradientStop[ ] gradientStops)

  Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.
- LinearGradientBrush RelativeTo (Graphics referenceGraphics)

  Returns a LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the referenceGraphics . This involves computing the current transform matrix of the referenceGraphics , inverting it, and applying the inverse matrix to the StartPoint and EndPoint of the current instance.
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Properties

- Point StartPoint `[get]`

  The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
- Point EndPoint `[get]`

  The end point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Additional Inherited Members

7.89.1 Detailed Description

Represents a brush painting with a linear gradient.

Definition at line 283 of file Brush.cs.

7.89.2 Constructor & Destructor Documentation

7.89.2.1 LinearGradientBrush() [1/2]

```
VectSharp.LinearGradientBrush.LinearGradientBrush (
            Point startPoint,
            Point endPoint,
            IEnumerable< GradientStop > gradientStops )
```

Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.

Parameters

startPoint	The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
endPoint	The ending point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 301 of file Brush.cs.

7.89.2.2 LinearGradientBrush() [2/2]

```
VectSharp.LinearGradientBrush.LinearGradientBrush (
            Point startPoint,
            Point endPoint,
            params GradientStop[] gradientStops )
```

Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.

Parameters

startPoint	The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
endPoint	The ending point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 315 of file Brush.cs.

7.89.3 Member Function Documentation

7.89.3.1 MultiplyOpacity()

```
override Brush VectSharp.LinearGradientBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 397 of file Brush.cs.

7.89.3.2 RelativeTo()

```
LinearGradientBrush VectSharp.LinearGradientBrush.RelativeTo (
            Graphics referenceGraphics )
```

Returns a LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the *referenceGraphics* . This involves computing the current transform matrix of the *referenceGraphics* , inverting it, and applying the inverse matrix to the StartPoint and EndPoint of the current instance.

Parameters

referenceGraphics	The Graphics whose original reference frame is to be used.

Returns

A LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the *referenceGraphics* .

Definition at line 347 of file Brush.cs.

7.89.4 Property Documentation

7.89.4.1 EndPoint

```
Point VectSharp.LinearGradientBrush.EndPoint  [get]
```

The end point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Definition at line 293 of file Brush.cs.

7.89.4.2 StartPoint

```
Point VectSharp.LinearGradientBrush.StartPoint  [get]
```

The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Definition at line 288 of file Brush.cs.

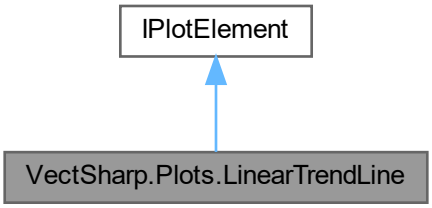
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.90 VectSharp.Plots.LinearTrendLine Class Reference

A plot element that draws a linear trendline with equation `y = a * x + b`.

Inheritance diagram for VectSharp.Plots.LinearTrendLine:



Public Member Functions

- LinearTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LinearTrendLine instance, specifying the equation parameters.
- LinearTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LinearTrendLine instance, determining the equation parameters by running a regression.
- LinearTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LinearTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope [get, set]

  The slope of the trendline (a).
- double Intercept [get, set]

  The intercept of the trendline (b).
- double MinX [get, set]

  The minimum X value for which the trendline is plotted.
- double MinY [get, set]

  The minimum Y value for which the trendline is plotted.
- double MaxX [get, set]

  The maximum X value for which the trendline is plotted.
- double MaxY [get, set]

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } [get, set]

  Presentation attributes for the trendline.
- string Tag [get, set]

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.

7.90.1 Detailed Description

A plot element that draws a linear trendline with equation `y = a * x + b`.

Definition at line 31 of file Trendlines.cs.

7.90.2 Constructor & Destructor Documentation

7.90.2.1 LinearTrendLine() [1/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LinearTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 89 of file Trendlines.cs.

7.90.2.2 LinearTrendLine() [2/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LinearTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 106 of file Trendlines.cs.

7.90.2.3 LinearTrendLine() [3/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LinearTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 154 of file Trendlines.cs.

7.90.3 Member Function Documentation

7.90.3.1 Plot()

```
void VectSharp.Plots.LinearTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 157 of file Trendlines.cs.

7.90.4 Property Documentation

7.90.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.LinearTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 76 of file Trendlines.cs.

7.90.4.2 Intercept

```
double VectSharp.Plots.LinearTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 41 of file Trendlines.cs.

7.90.4.3 MaxX

```
double VectSharp.Plots.LinearTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 56 of file Trendlines.cs.

7.90.4.4 MaxY

```
double VectSharp.Plots.LinearTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 61 of file Trendlines.cs.

7.90.4.5 MinX

```
double VectSharp.Plots.LinearTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 46 of file Trendlines.cs.

7.90.4.6 MinY

```
double VectSharp.Plots.LinearTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 51 of file Trendlines.cs.

7.90.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.LinearTrendLine.PresentationAttributes = new
PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 66 of file Trendlines.cs.

7.90.4.8 Slope

```
double VectSharp.Plots.LinearTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 36 of file Trendlines.cs.

7.90.4.9 Tag

```
string VectSharp.Plots.LinearTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 71 of file Trendlines.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.91 VectSharp.LineDash Struct Reference

Represents instructions on how to paint a dashed line.

Public Member Functions

- LineDash (double unitsOn, double unitsOff, double phase)
  Define a new line dash pattern.

Public Attributes

- double UnitsOn
  Length of the "on" (painted) segment.
- double UnitsOff
  Length of the "off" (not painted) segment.
- double Phase
  Position in the dash pattern at which the line starts.

Static Public Attributes

- static LineDash SolidLine = new LineDash(0, 0, 0)
  A solid (not dashed) line

7.91.1 Detailed Description

Represents instructions on how to paint a dashed line.

Definition at line 112 of file Enums.cs.

7.91.2 Constructor & Destructor Documentation

7.91.2.1 LineDash()

```
VectSharp.LineDash.LineDash (
            double unitsOn,
            double unitsOff,
            double phase )
```

Define a new line dash pattern.

Parameters

unitsOn	The length of the "on" (painted) segment.
unitsOff	The length of the "off" (not painted) segment.
phase	The position in the dash pattern at which the line starts.

Definition at line 140 of file Enums.cs.

7.91.3 Member Data Documentation

7.91.3.1 Phase

```
double VectSharp.LineDash.Phase
```

Position in the dash pattern at which the line starts.

Definition at line 132 of file Enums.cs.

7.91.3.2 SolidLine

```
LineDash VectSharp.LineDash.SolidLine = new LineDash(0, 0, 0)  [static]
```

A solid (not dashed) line

Definition at line 117 of file Enums.cs.

7.91.3.3 UnitsOff

```
double VectSharp.LineDash.UnitsOff
```

Length of the "off" (not painted) segment.

Definition at line 127 of file Enums.cs.

7.91.3.4 UnitsOn

```
double VectSharp.LineDash.UnitsOn
```

Length of the "on" (painted) segment.

Definition at line 122 of file Enums.cs.

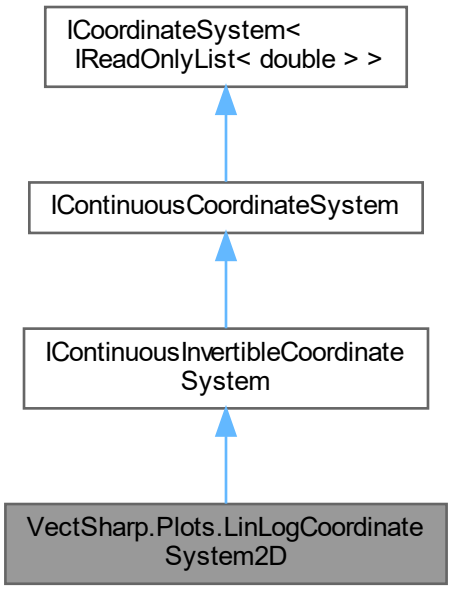
The documentation for this struct was generated from the following file:

- VectSharp/Enums.cs

7.92 VectSharp.Plots.LinLogCoordinateSystem2D Class Reference

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.

Inheritance diagram for VectSharp.Plots.LinLogCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LinLogCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LinLogCoordinateSystem2D manually specifying the parameter values.

- LinLogCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LinLogCoordinateSystem2D determining the value range from the data .

- LinLogCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LinLogCoordinateSystem2D determining the value range from the data .

- double[ ] ToDataCoordinates (Point plotPoint)

Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.92.1 Detailed Description

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.

Definition at line 719 of file CoordinateSystems.cs.

7.92.2 Constructor & Destructor Documentation

7.92.2.1 LinLogCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LinLogCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 799 of file CoordinateSystems.cs.

7.92.2.2 LinLogCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinLogCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 815 of file CoordinateSystems.cs.

7.92.2.3 LinLogCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinLogCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 856 of file CoordinateSystems.cs.

7.92.3 Member Function Documentation

7.92.3.1 GetAround()

```
double[] VectSharp.Plots.LinLogCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 914 of file CoordinateSystems.cs.

7.92.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LinLogCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 755 of file CoordinateSystems.cs.

7.92.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LinLogCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 892 of file CoordinateSystems.cs.

7.92.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LinLogCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 898 of file CoordinateSystems.cs.

7.92.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LinLogCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 908 of file CoordinateSystems.cs.

7.92.4 Property Documentation

7.92.4.1 IsLinear

```
bool VectSharp.Plots.LinLogCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 752 of file CoordinateSystems.cs.

7.92.4.2 MaxX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 729 of file CoordinateSystems.cs.

7.92.4.3 MaxY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 739 of file CoordinateSystems.cs.

7.92.4.4 MinX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value.

Definition at line 724 of file CoordinateSystems.cs.

7.92.4.5 MinY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 734 of file CoordinateSystems.cs.

7.92.4.6 Resolution

```
double [] VectSharp.Plots.LinLogCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 770 of file CoordinateSystems.cs.

7.92.4.7 ScaleX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 744 of file CoordinateSystems.cs.

7.92.4.8 ScaleY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 749 of file CoordinateSystems.cs.

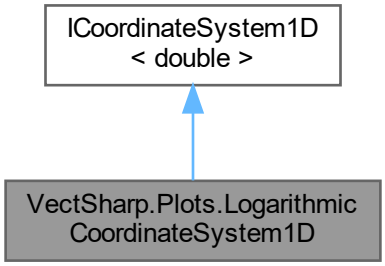
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.93 VectSharp.Plots.LogarithmicCoordinateSystem1D Class Reference

Represents a 1-D logarithmic coordinate system.

Inheritance diagram for VectSharp.Plots.LogarithmicCoordinateSystem1D:



Public Member Functions

- LogarithmicCoordinateSystem1D (double min, double max, double scale)

  Creates a new LogarithmicCoordinateSystem1D, manually specifying the parameters.
- LogarithmicCoordinateSystem1D (IReadOnlyList< double > data, double scale=350)

  Creates a new LogarithmicCoordinateSystem1D determining the value range from the data .
- double ToPlotCoordinates (double dataPoint)

Properties

- double Min `[get, set]`

  The minimum value (in logarithmic scale).
- double Max `[get, set]`

  The maximum value (in logarithmic scale).
- double Scale `[get, set]`

  The scale factor.

7.93.1 Detailed Description

Represents a 1-D logarithmic coordinate system.

Definition at line 987 of file CoordinateSystems.cs.

7.93.2 Constructor & Destructor Documentation

7.93.2.1 LogarithmicCoordinateSystem1D() [1/2]

```
VectSharp.Plots.LogarithmicCoordinateSystem1D.LogarithmicCoordinateSystem1D (
            double min,
            double max,
            double scale )
```

Creates a new LogarithmicCoordinateSystem1D, manually specifying the parameters.

Parameters

min	The minimum value.
max	The maximum value.
scale	The scale factor.

Definition at line 1010 of file CoordinateSystems.cs.

7.93.2.2 LogarithmicCoordinateSystem1D() [2/2]

```
VectSharp.Plots.LogarithmicCoordinateSystem1D.LogarithmicCoordinateSystem1D (
            IReadOnlyList< double > data,
            double scale = 350 )
```

Creates a new LogarithmicCoordinateSystem1D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scale	The scale factor.

Definition at line 1022 of file CoordinateSystems.cs.

7.93.3 Member Function Documentation

7.93.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.ToPlotCoordinates (
            double dataPoint )
```

Definition at line 1045 of file CoordinateSystems.cs.

7.93.4 Property Documentation

7.93.4.1 Max

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Max  [get], [set]
```

The maximum value (in logarithmic scale).

Definition at line 997 of file CoordinateSystems.cs.

7.93.4.2 Min

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Min  [get], [set]
```

The minimum value (in logarithmic scale).

Definition at line 992 of file CoordinateSystems.cs.

7.93.4.3 Scale

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Scale  [get], [set]
```

The scale factor.

Definition at line 1002 of file CoordinateSystems.cs.

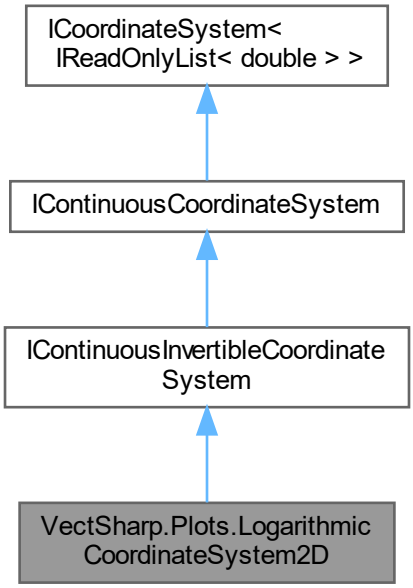
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.94 VectSharp.Plots.LogarithmicCoordinateSystem2D Class Reference

Represents a logarithmic coordinate system.

Inheritance diagram for VectSharp.Plots.LogarithmicCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LogarithmicCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LogarithmicCoordinateSystem2D manually specifying the parameter values.

- LogarithmicCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LogarithmicCoordinateSystem2D determining the value range from the data .

- LogarithmicCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LogarithmicCoordinateSystem2D determining the value range from the data .

- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value (in logarithmic scale).
- double MaxX `[get, set]`

  The maximum X value (in logarithmic scale).
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.94.1 Detailed Description

Represents a logarithmic coordinate system.

Definition at line 310 of file CoordinateSystems.cs.

7.94.2 Constructor & Destructor Documentation

7.94.2.1 LogarithmicCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LogarithmicCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 390 of file CoordinateSystems.cs.

7.94.2.2 LogarithmicCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogarithmicCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 406 of file CoordinateSystems.cs.

7.94.2.3 LogarithmicCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogarithmicCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 448 of file CoordinateSystems.cs.

7.94.3 Member Function Documentation

7.94.3.1 GetAround()

```
double[] VectSharp.Plots.LogarithmicCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 506 of file CoordinateSystems.cs.

7.94.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LogarithmicCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 346 of file CoordinateSystems.cs.

7.94.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LogarithmicCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 484 of file CoordinateSystems.cs.

7.94.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LogarithmicCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 490 of file CoordinateSystems.cs.

7.94.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LogarithmicCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 500 of file CoordinateSystems.cs.

7.94.4 Property Documentation

7.94.4.1 IsLinear

```
bool VectSharp.Plots.LogarithmicCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 343 of file CoordinateSystems.cs.

7.94.4.2 MaxX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value (in logarithmic scale).

Definition at line 320 of file CoordinateSystems.cs.

7.94.4.3 MaxY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 330 of file CoordinateSystems.cs.

7.94.4.4 MinX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value (in logarithmic scale).

Definition at line 315 of file CoordinateSystems.cs.

7.94.4.5 MinY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 325 of file CoordinateSystems.cs.

7.94.4.6 Resolution

```
double [] VectSharp.Plots.LogarithmicCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 361 of file CoordinateSystems.cs.

7.94.4.7 ScaleX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 335 of file CoordinateSystems.cs.

7.94.4.8 ScaleY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 340 of file CoordinateSystems.cs.

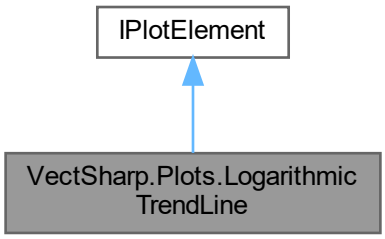
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.95 VectSharp.Plots.LogarithmicTrendLine Class Reference

A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.

Inheritance diagram for VectSharp.Plots.LogarithmicTrendLine:



Public Member Functions

- LogarithmicTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LogarithmicTrendLine instance, specifying the equation parameters.
- LogarithmicTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.
- LogarithmicTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩ System, double? fixedIntercept=null)

  Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope `[get, set]`

  The slope of the trendline (a).
- double Intercept `[get, set]`

  The intercept of the trendline (b).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

The maximum X value for which the trendline is plotted.

- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.

- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.

- string Tag `[get, set]`

  A tag to identify the trendline in the plot.

- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.95.1 Detailed Description

A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.

Definition at line 474 of file Trendlines.cs.

7.95.2 Constructor & Destructor Documentation

7.95.2.1 LogarithmicTrendLine() [1/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LogarithmicTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 532 of file Trendlines.cs.

7.95.2.2 LogarithmicTrendLine() [2/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 549 of file Trendlines.cs.

7.95.2.3 LogarithmicTrendLine() [3/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 596 of file Trendlines.cs.

7.95.3 Member Function Documentation

7.95.3.1 Plot()

```
void VectSharp.Plots.LogarithmicTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 599 of file Trendlines.cs.

7.95.4 Property Documentation

7.95.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.LogarithmicTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 519 of file Trendlines.cs.

7.95.4.2 Intercept

```
double VectSharp.Plots.LogarithmicTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 484 of file Trendlines.cs.

7.95.4.3 MaxX

```
double VectSharp.Plots.LogarithmicTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 499 of file Trendlines.cs.

7.95.4.4 MaxY

```
double VectSharp.Plots.LogarithmicTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 504 of file Trendlines.cs.

7.95.4.5 MinX

```
double VectSharp.Plots.LogarithmicTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 489 of file Trendlines.cs.

7.95.4.6 MinY

```
double VectSharp.Plots.LogarithmicTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 494 of file Trendlines.cs.

7.95.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.LogarithmicTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 509 of file Trendlines.cs.

7.95.4.8 Slope

```
double VectSharp.Plots.LogarithmicTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 479 of file Trendlines.cs.

7.95.4.9 Tag

```
string VectSharp.Plots.LogarithmicTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 514 of file Trendlines.cs.

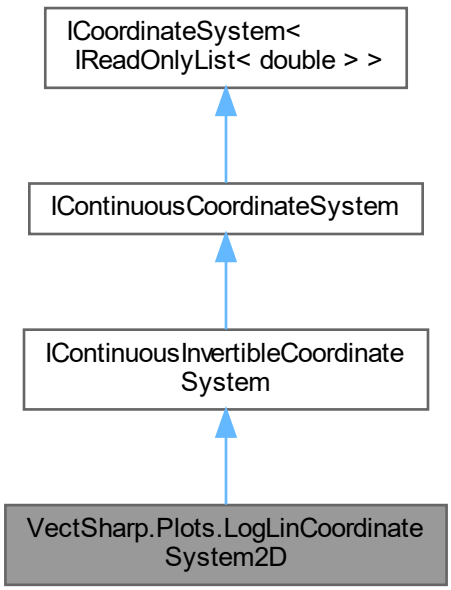
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.96 VectSharp.Plots.LogLinCoordinateSystem2D Class Reference

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.

Inheritance diagram for VectSharp.Plots.LogLinCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` if any two points aligned along direction in data space are also aligned along some line in plot space, `false` otherwise.

- LogLinCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LogLinCoordinateSystem2D manually specifying the parameter values.
- LogLinCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LogLinCoordinateSystem2D determining the value range from the data .
- LogLinCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LogLinCoordinateSystem2D determining the value range from the data .
- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.96.1 Detailed Description

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.

Definition at line 515 of file CoordinateSystems.cs.

7.96.2 Constructor & Destructor Documentation

7.96.2.1 LogLinCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LogLinCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 595 of file CoordinateSystems.cs.

7.96.2.2 LogLinCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogLinCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 611 of file CoordinateSystems.cs.

7.96.2.3 LogLinCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogLinCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 652 of file CoordinateSystems.cs.

7.96.3 Member Function Documentation

7.96.3.1 GetAround()

```
double[] VectSharp.Plots.LogLinCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 710 of file CoordinateSystems.cs.

7.96.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LogLinCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 551 of file CoordinateSystems.cs.

7.96.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LogLinCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 688 of file CoordinateSystems.cs.

7.96.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LogLinCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 694 of file CoordinateSystems.cs.

7.96.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LogLinCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 704 of file CoordinateSystems.cs.

7.96.4 Property Documentation

7.96.4.1 IsLinear

```
bool VectSharp.Plots.LogLinCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 548 of file CoordinateSystems.cs.

7.96.4.2 MaxX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 525 of file CoordinateSystems.cs.

7.96.4.3 MaxY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 535 of file CoordinateSystems.cs.

7.96.4.4 MinX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value.

Definition at line 520 of file CoordinateSystems.cs.

7.96.4.5 MinY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 530 of file CoordinateSystems.cs.

7.96.4.6 Resolution

```
double [] VectSharp.Plots.LogLinCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 566 of file CoordinateSystems.cs.

7.96.4.7 ScaleX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 540 of file CoordinateSystems.cs.

7.96.4.8 ScaleY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 545 of file CoordinateSystems.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.97 VectSharp.Markdown.Margins Class Reference

Represents the margins of a page.

Public Member Functions

- Margins (double left, double top, double right, double bottom)

  Creates a new Margins instance.

Properties

- double Left `[get]`

  The left margin.
- double Right `[get]`

  The right margin.
- double Top `[get]`

  The top margin.
- double Bottom `[get]`

  The bottom margin.

7.97.1 Detailed Description

Represents the margins of a page.

Definition at line 185 of file MarkdownContext.cs.

7.97.2 Constructor & Destructor Documentation

7.97.2.1 Margins()

```
VectSharp.Markdown.Margins.Margins (
            double left,
            double top,
            double right,
            double bottom )
```

Creates a new Margins instance.

Parameters

left	The left margin.
top	The top margin.
right	The right margin.
bottom	The bottom margin.

Definition at line 214 of file MarkdownContext.cs.

7.97.3 Property Documentation

7.97.3.1 Bottom

```
double VectSharp.Markdown.Margins.Bottom  [get]
```

The bottom margin.

Definition at line 205 of file MarkdownContext.cs.

7.97.3.2 Left

```
double VectSharp.Markdown.Margins.Left  [get]
```

The left margin.

Definition at line 190 of file MarkdownContext.cs.

7.97.3.3 Right

```
double VectSharp.Markdown.Margins.Right  [get]
```

The right margin.

Definition at line 195 of file MarkdownContext.cs.

7.97.3.4 Top

```
double VectSharp.Markdown.Margins.Top  [get]
```

The top margin.

Definition at line 200 of file MarkdownContext.cs.

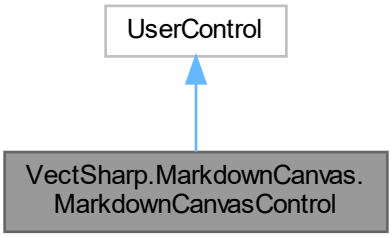
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownContext.cs

7.98 VectSharp.MarkdownCanvas.MarkdownCanvasControl Class Reference

A control to display a Markdown document in an Avalonia application.

Inheritance diagram for VectSharp.MarkdownCanvas.MarkdownCanvasControl:



Public Member Functions

- MarkdownCanvasControl ()

  Initialises a new MarkdownCanvasControl.

Static Public Attributes

- static readonly StyledProperty< double > MaxRenderWidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth), double.PositiveInfinity)

  Defines the MaxRenderWidth property.
- static readonly StyledProperty< double > MinRenderWidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth), 200)

  Defines the MinRenderWidth property.
- static readonly StyledProperty< double > MinVariationProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10)

  Defines the MinVariation property.
- static readonly StyledProperty< string > DocumentSourceProperty = AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource))

  Defines the DocumentSource property.
- static readonly StyledProperty< MarkdownDocument > DocumentProperty = AvaloniaProperty.↩ Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document))

  Defines the Document property.
- static readonly StyledProperty< AvaloniaContextInterpreter.TextOptions > TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl, AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOptior AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary)

  Defines the TextConversionOption property.

Properties

- double MaxRenderWidth [get, set]

  The maximum width for the rendered document. This will be used even if the control's client area is larger than this (the alignment of the document within the controll will depend on the control's ContentControl.HorizontalContent↩ Alignment).
- double MinRenderWidth [get, set]

  The minimum width for the rendered document. If the control's client area is smaller than this, the horizontal scroll bar will be activated.
- double MinVariation [get, set]

  The minimum width variation that triggers a document reflow. If the control is resized, but the width changes by less than this amount, the document is not re-drawn.
- string DocumentSource [set]

  Sets the currently displayed document from Markdown source.
- MarkdownDocument Document [get, set]

  Gets or sets the currently displayed MarkdownDocument.
- AvaloniaContextInterpreter.TextOptions TextConversionOption [get, set]

  Gets or sets the value that determines whether text items should be converted into paths when drawing. Setting this to AvaloniaContextInterpreter.TextOptions.NeverConvert will improve performance if you are using custom fonts, but may cause unexpected results unless the font families being used are of type ResourceFontFamily.
- MarkdownRenderer Renderer [get]

  The MarkdownRenderer used to render the Document. You can use the properties of this object to customise the rendering. Note that setting the Avalonia.Controls.Primitives.TemplatedControl.FontSize of the MarkdownCanvasControl will propagate to the Renderer's MarkdownRenderer.BaseFontSize.

7.98.1 Detailed Description

A control to display a Markdown document in an Avalonia application.

Definition at line 36 of file MarkdownCanvas.axaml.cs.

7.98.2 Constructor & Destructor Documentation

7.98.2.1 MarkdownCanvasControl()

```
VectSharp.MarkdownCanvas.MarkdownCanvasControl.MarkdownCanvasControl ( )
```

Initialises a new MarkdownCanvasControl.

Definition at line 134 of file MarkdownCanvas.axaml.cs.

7.98.3 Member Data Documentation

7.98.3.1 DocumentProperty

```
readonly StyledProperty<MarkdownDocument> VectSharp.MarkdownCanvas.MarkdownCanvasControl.↩
DocumentProperty = AvaloniaProperty.Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document))
[static]
```

Defines the Document property.

Definition at line 96 of file MarkdownCanvas.axaml.cs.

7.98.3.2 DocumentSourceProperty

```
readonly StyledProperty<string> VectSharp.MarkdownCanvas.MarkdownCanvasControl.Document↩
SourceProperty = AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource))
[static]
```

Defines the DocumentSource property.

Definition at line 83 of file MarkdownCanvas.axaml.cs.

7.98.3.3 MaxRenderWidthProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MaxRender↩
WidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth),
double.PositiveInfinity)  [static]
```

Defines the MaxRenderWidth property.

Definition at line 41 of file MarkdownCanvas.axaml.cs.

7.98.3.4 MinRenderWidthProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinRender↩
WidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth),
200)  [static]
```

Defines the MinRenderWidth property.

Definition at line 55 of file MarkdownCanvas.axaml.cs.

7.98.3.5 MinVariationProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinVariation↩
Property = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10)
[static]
```

Defines the MinVariation property.

Definition at line 69 of file MarkdownCanvas.axaml.cs.

7.98.3.6 TextConversionOptionsProperty

```
readonly StyledProperty<AvaloniaContextInterpreter.TextOptions> VectSharp.MarkdownCanvas.↩
MarkdownCanvasControl.TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl,
AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOption), AvaloniaContextInterpreter.↩
TextOptions.ConvertIfNecessary)  [static]
```

Defines the TextConversionOption property.

Definition at line 110 of file MarkdownCanvas.axaml.cs.

7.98.4 Property Documentation

7.98.4.1 Document

```
MarkdownDocument VectSharp.MarkdownCanvas.MarkdownCanvasControl.Document  [get], [set]
```

Gets or sets the currently displayed MarkdownDocument.

Definition at line 101 of file MarkdownCanvas.axaml.cs.

7.98.4.2 DocumentSource

```
string VectSharp.MarkdownCanvas.MarkdownCanvasControl.DocumentSource  [set]
```

Sets the currently displayed document from Markdown source.

Definition at line 88 of file MarkdownCanvas.axaml.cs.

7.98.4.3 MaxRenderWidth

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MaxRenderWidth [get], [set]
```

The maximum width for the rendered document. This will be used even if the control's client area is larger than this (the alignment of the document within the controll will depend on the control's ContentControl.HorizontalContent↩ Alignment).

Definition at line 46 of file MarkdownCanvas.axaml.cs.

7.98.4.4 MinRenderWidth

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinRenderWidth [get], [set]
```

The minimum width for the rendered document. If the control's client area is smaller than this, the horizontal scroll bar will be activated.

Definition at line 60 of file MarkdownCanvas.axaml.cs.

7.98.4.5 MinVariation

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinVariation [get], [set]
```

The minimum width variation that triggers a document reflow. If the control is resized, but the width changes by less than this amount, the document is not re-drawn.

Definition at line 74 of file MarkdownCanvas.axaml.cs.

7.98.4.6 Renderer

```
MarkdownRenderer VectSharp.MarkdownCanvas.MarkdownCanvasControl.Renderer [get]
```

The MarkdownRenderer used to render the Document. You can use the properties of this object to customise the rendering. Note that setting the Avalonia.Controls.Primitives.TemplatedControl.FontSize of the MarkdownCanvasControl will propagate to the Renderer's MarkdownRenderer.BaseFontSize.

Definition at line 125 of file MarkdownCanvas.axaml.cs.

7.98.4.7 TextConversionOption

```
AvaloniaContextInterpreter.TextOptions VectSharp.MarkdownCanvas.MarkdownCanvasControl.Text↩
ConversionOption [get], [set]
```

Gets or sets the value that determines whether text items should be converted into paths when drawing. Setting this to AvaloniaContextInterpreter.TextOptions.NeverConvert will improve performance if you are using custom fonts, but may cause unexpected results unless the font families being used are of type ResourceFontFamily.

Definition at line 116 of file MarkdownCanvas.axaml.cs.

The documentation for this class was generated from the following file:

- VectSharp.MarkdownCanvas/MarkdownCanvas.axaml.cs

7.99 VectSharp.Markdown.MarkdownRenderer Class Reference

Renders Markdown documents into VectSharp graphics objects.

Public Types

- enum VerticalAlignment

  Defines the options for the vertical alignment of table cells.

Public Member Functions

- Page RenderSinglePage (string markdownSource, double width, out Dictionary< string, string > link↩Destinations)

  Parses the supplied markdownSource using all the supported extensions and renders the resulting document. Page breaks are disabled, and the document is rendered as a single page with the specified width . The page will be cropped at the appropriate height to contain the entire document.
- Page RenderSinglePage (MarkdownDocument markdownDocument, double width, out Dictionary< string, string > linkDestinations)

  Renders the supplied markdownDocument . Page breaks are disabled, and the document is rendered as a single page with the specified width . The page will be cropped at the appropriate height to contain the entire document.
- Document Render (string markdownSource, out Dictionary< string, string > linkDestinations)

  Parses the supplied markdownSource using all the supported extensions and renders the resulting document. The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.
- Document Render (MarkdownDocument mardownDocument, out Dictionary< string, string > link↩Destinations)

  Renders the supplied mardownDocument . The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Properties

- double BaseFontSize = 9.71424 `[get, set]`

  The base font size to use when rendering the document. This will be the size of regular elements, and the size of header elements will be expressed as a multiple of this.
- double MathFontScalingFactor = 0.85 `[get, set]`

  Scaling factor for the font size to use when rendering math.
- double[ ] HeaderFontSizeMultipliers `[get]`

  The font size for elements at each header level. The values in this array will be multiplied by the BaseFontSize.
- double[ ] HeaderLineThicknesses = new double[ ] { 1, 1, 0, 0, 0, 0 } `[get]`

  The thickness of the separator line after a header of each level. A value of 0 disables the line after headers of that level.
- double ThematicBreakThickness = 2 `[get, set]`

  The thickness of thematic break lines.
- FontFamily RegularFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica) `[get, set]`

  The font family for regular text.
- FontFamily BoldFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica↩Bold) `[get, set]`

  The font family for bold text.
- FontFamily ItalicFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica↩Oblique) `[get, set]`

  The font family for italic text.
- FontFamily BoldItalicFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.↩HelveticaBoldOblique) `[get, set]`

  The font family for bold italic text.
- FontFamily CodeFont = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier) `[get, set]`

  The font family for code elements.
- FontFamily CodeFontBold = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold) `[get, set]`

  The font family for bold code elements.
- FontFamily CodeFontItalic = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier↩Oblique) `[get, set]`

  The font family for italic code elements.
- FontFamily CodeFontBoldItalic = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier↩BoldOblique) `[get, set]`

  The font family for bold italic code elements.
- double UnderlineThickness = 0.075 `[get, set]`

  The thickness of underlines. This value will be multiplied by the font size of the element being underlined.
- double BoldUnderlineThickness = 0.15 `[get, set]`

  The thickness of underlines for bold text. This value will be multiplied by the font size of the element being underlined.
- Margins Margins = new Margins(55, 55, 55, 55) `[get, set]`

  The margins of the page.
- Margins TableCellMargins = new Margins(5, 0, 5, 0) `[get, set]`

  The margins for table cells.
- VerticalAlignment TableVAlign = VerticalAlignment.Middle `[get, set]`

  The vertical alignment of table cells.
- Size PageSize = new Size(595, 842) `[get, set]`

  The size of the page.
- double SpaceBeforeParagaph = 0 `[get, set]`

  The space before each text paragraph. This value will be multiplied by the BaseFontSize.

- double SpaceAfterParagraph = 0.75 [get, set]

  The space after each text paragraph. This value will be multiplied by the BaseFontSize.
- double SpaceAfterLine = 0.25 [get, set]

  The space after each line of text. This value will be multiplied by the BaseFontSize.
- double SpaceBeforeHeading = 0.25 [get, set]

  The space before each heading. This value will be multiplied by the font size of the heading.
- double SpaceAfterHeading = 0.25 [get, set]

  The space after each heading. This value will be multiplied by the font size of the heading.
- double CodeInlineMargin = 0.25 [get, set]

  The margin at the left and right of code inlines. This value will be multiplied by the current font size.
- double IndentWidth = 40 [get, set]

  The indentation width used for list items.
- double QuoteBlockIndentWidth = 30 [get, set]

  The indentation width used for block quotes.
- double QuoteBlockBarWidth = 5 [get, set]

  The thickness of the bar to the left of block quotes.
- double SubSuperscriptFontSize = 0.7 [get, set]

  The font size for subscripts and superscripts. This value will be multiplied by the current font size.
- double SuperscriptShift = 0.33 [get, set]

  The upwards shift in the baseline for superscript elements. This value will be multiplied by the current font size.
- double SubscriptShift = 0.14 [get, set]

  The downwards shift in the baseline for subscript elements. This value will be multiplied by the current font size.
- string BaseImageUri = "" [get, set]

  The base uri for resolving relative image addresses.
- Func< string, string,(string, bool)> ImageUriResolver = HTTPUtils.ResolveImageURI [get, set]

  A method used to resolve (possibly remote) image uris into local file paths. The first argument of the method should be the image uri and the second argument the base uri used to resolve relative links. The method should return a tuple containing the path of the local file and a boolean value indicating whether the file has been fetched from a remote location and should be deleted after the program has finished using it.
- Uri BaseLinkUri = new Uri("about:blank") [get, set]

  The base uri for resolving links.
- Func< string, string > LinkUriResolver = a => a [get, set]

  A method used to resolve link addresses. The argument of the method should be the absolute link, and the method should return the resolved address. This can be used to "redirect" links to a different target.
- Func< string, RasterImage > RasterImageLoader = null [get, set]

  A method used to a load raster image from a local file. The argument of the method should be the path of a local image file, and the method should return a RasterImage representing that file. For example, this can be achieved using the `RasterImageFile` class from the `VectSharp.MuPDFUtils` package. If this is `null`, only SVG images will be included in the document.
- double ImageUnitMultiplier = 0.60714 [get, set]

  The size of images (as defined in the image's width and height attributes) will be multiplied by this value to determine the actual size of the image on the page. This has no effect on images without a width or height attribute.
- double ImageMultiplier = 1 [get, set]

  The size of images will be multiplied by this value to determine the actual size of the image on the page. For images that have a width or height attribute, this will be applied in addition to the ImageUnitMultiplier. For images without width and height, only this multiplier will be applied.
- double ImageSideMargin = 10 [get, set]

  The margin on the right of left-aligned images and on the left of right-aligned images.
- double ImageMarginTolerance = 25 [get, set]

  Images will be allowed to extend into the page bottom margin area by this amount before triggering a page break. This should be smaller than the bottom margin, otherwise images risk being cut off by the page boundary.
- Func< string, string, List< List< FormattedString > > > SyntaxHighlighter = VectSharp.Markdown.SyntaxHighlighter.GetSynta
  [get, set]

A method used for syntax highlighting. The first argument should be the source code to highlight, while the second parameter is the name of the language to use for the highlight. The method should return a list of lists of FormattedStrings, with each list of FormattedStrings representing a line. For each code block, if the method returns `null`, no syntax highlighting is used.

- List< Action< Graphics, Colour > > Bullets `[get]`

  Bullet points used for unordered lists. Each element of this list corresponds to the bullet for each level of list indentation. If the list indentation is greater than the number of elements in this list, the bullet points will be reused cyclically. Each element of this list is a method taking two arguments: the first is the Graphics object on which the bullet point should be drawn, while the second is the colour in which it should be painted. The method should draw the bullet point centered around the origin. The size of the bullet point will be multiplied by the current font size.
- Colour ForegroundColour = Colours.Black `[get, set]`

  The foreground colour for text elements.
- Colour BackgroundColour = Colours.White `[get, set]`

  The background colour for the page.
- Colour HeaderLineColour = Colour.FromRgb(180, 180, 180) `[get, set]`

  The colour of the line below headers.
- Colour ThematicBreakLineColour = Colour.FromRgb(180, 180, 200) `[get, set]`

  The colour for thematic break lines.
- Colour LinkColour = Colour.FromRgb(25, 140, 191) `[get, set]`

  The colour for hypertext links-
- Colour CodeInlineBackgroundColour = Colour.FromRgb(240, 240, 240) `[get, set]`

  The background colour for code inlines.
- Colour CodeBlockBackgroundColour = Colour.FromRgb(240, 240, 245) `[get, set]`

  The background colour for code blocks.
- Colour QuoteBlockBarColour = Colour.FromRgb(75, 152, 220) `[get, set]`

  The colour for the bar to the left of block quotes.
- Colour QuoteBlockBackgroundColour = Colour.FromRgb(240, 240, 255) `[get, set]`

  The background colour for block quotes.
- Colour InsertedColour = Colour.FromRgb(0, 158, 115) `[get, set]`

  The colour for text that has been styled as "inserted".
- Colour MarkedColour = Colour.FromRgb(213, 94, 0) `[get, set]`

  The colour for text that has been styled as "marked".
- Colour TableHeaderRowSeparatorColour = Colours.Black `[get, set]`

  The colour for the line separating the table header row from normal rows.
- Colour TableRowSeparatorColour = Colour.FromRgb(180, 180, 180) `[get, set]`

  The colour for lines separating table rows from each other.
- double TableHeaderRowSeparatorThickness = 2 `[get, set]`

  The thickness of the line separating the table header row from normal rows.
- double TableHeaderSeparatorThickness = 1 `[get, set]`

  The thickness of lines separating table rows from each other.
- Graphics TaskListUncheckedBullet `[get, set]`

  The bullet used for unchecked task list items.
- Graphics TaskListCheckedBullet `[get, set]`

  The bullet used for checked task list items.
- bool AllowPageBreak = true `[get, set]`

  Determines whether page breaks should be treated as such in the source.

7.99.1 Detailed Description

Renders Markdown documents into VectSharp graphics objects.

Definition at line 37 of file MarkdownRenderer.cs.

7.99.2 Member Enumeration Documentation

7.99.2.1 VerticalAlignment

```
enum VectSharp.Markdown.MarkdownRenderer.VerticalAlignment
```

Defines the options for the vertical alignment of table cells.

Definition at line 130 of file MarkdownRenderer.cs.

7.99.3 Member Function Documentation

7.99.3.1 Render() [1/2]

```
Document VectSharp.Markdown.MarkdownRenderer.Render (
            MarkdownDocument mardownDocument,
            out Dictionary< string, string > linkDestinations )
```

Renders the supplied *mardownDocument* . The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Parameters

mardownDocument	The markdown document to render.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Document to a file.

Returns

A Document containing a rendering of the supplied markdown document, consisting of one or more pages of the size specified in the PageSize of the current instance.

Definition at line 504 of file MarkdownRenderer.cs.

7.99.3.2 Render() [2/2]

```
Document VectSharp.Markdown.MarkdownRenderer.Render (
            string markdownSource,
            out Dictionary< string, string > linkDestinations )
```

Parses the supplied *markdownSource* using all the supported extensions and renders the resulting document. The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Parameters

markdownSource	The markdown source to parse.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Document to a file.

Returns

A Document containing a rendering of the supplied markdown document, consisting of one or more pages of the size specified in the PageSize of the current instance.

Definition at line 491 of file MarkdownRenderer.cs.

7.99.3.3 RenderSinglePage() [1/2]

```
Page VectSharp.Markdown.MarkdownRenderer.RenderSinglePage (
            MarkdownDocument markdownDocument,
            double width,
            out Dictionary< string, string > linkDestinations )
```

Renders the supplied *markdownDocument* . Page breaks are disabled, and the document is rendered as a single page with the specified *width* . The page will be cropped at the appropriate height to contain the entire document.

Parameters

markdownDocument	The markdown document to render.
width	The width of the page.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.

Returns

A Page containing a rendering of the supplied markdown document.

Definition at line 432 of file MarkdownRenderer.cs.

7.99.3.4 RenderSinglePage() [2/2]

```
Page VectSharp.Markdown.MarkdownRenderer.RenderSinglePage (
            string markdownSource,
            double width,
            out Dictionary< string, string > linkDestinations )
```

Parses the supplied *markdownSource* using all the supported extensions and renders the resulting document. Page breaks are disabled, and the document is rendered as a single page with the specified *width* . The page will be cropped at the appropriate height to contain the entire document.

Parameters

markdownSource	The markdown source to parse.
width	The width of the page.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.

Returns

A Page containing a rendering of the supplied markdown document.

Definition at line 418 of file MarkdownRenderer.cs.

7.99.4 Property Documentation

7.99.4.1 AllowPageBreak

```
bool VectSharp.Markdown.MarkdownRenderer.AllowPageBreak = true  [get], [set]
```

Determines whether page breaks should be treated as such in the source.

Definition at line 402 of file MarkdownRenderer.cs.

7.99.4.2 BackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.BackgroundColour = Colours.White  [get], [set]
```

The background colour for the page.

Definition at line 298 of file MarkdownRenderer.cs.

7.99.4.3 BaseFontSize

```
double VectSharp.Markdown.MarkdownRenderer.BaseFontSize = 9.71424  [get], [set]
```

The base font size to use when rendering the document. This will be the size of regular elements, and the size of header elements will be expressed as a multiple of this.

Definition at line 42 of file MarkdownRenderer.cs.

7.99.4.4 BaseImageUri

```
string VectSharp.Markdown.MarkdownRenderer.BaseImageUri = ""  [get], [set]
```

The base uri for resolving relative image addresses.

Definition at line 221 of file MarkdownRenderer.cs.

7.99.4.5 BaseLinkUri

```
Uri VectSharp.Markdown.MarkdownRenderer.BaseLinkUri = new Uri("about:blank")  [get], [set]
```

The base uri for resolving links.

Definition at line 231 of file MarkdownRenderer.cs.

7.99.4.6 BoldFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.BoldFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBold)  [get], [set]
```

The font family for bold text.

Definition at line 75 of file MarkdownRenderer.cs.

7.99.4.7 BoldItalicFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.BoldItalicFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBoldOblique)  [get], [set]
```

The font family for bold italic text.

Definition at line 85 of file MarkdownRenderer.cs.

7.99.4.8 BoldUnderlineThickness

```
double VectSharp.Markdown.MarkdownRenderer.BoldUnderlineThickness = 0.15  [get], [set]
```

The thickness of underlines for bold text. This value will be multiplied by the font size of the element being underlined.

Definition at line 115 of file MarkdownRenderer.cs.

7.99.4.9 Bullets

```
List<Action<Graphics, Colour> > VectSharp.Markdown.MarkdownRenderer.Bullets  [get]
```

Initial value:

```
= new List<Action<Graphics, Colour»()
        {
            (graphics, colour) =>
            {
                graphics.FillPath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour);
            },
            (graphics, colour) =>
            {
                graphics.StrokePath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour, 0.1);
            },
            (graphics, colour) =>
            {
                graphics.StrokeRectangle(-0.75, -0.25, 0.5, 0.5, colour, 0.1);
            },
        }
```

Bullet points used for unordered lists. Each element of this list corresponds to the bullet for each level of list indentation. If the list indentation is greater than the number of elements in this list, the bullet points will be reused cyclically. Each element of this list is a method taking two arguments: the first is the Graphics object on which the bullet point should be drawn, while the second is the colour in which it should be painted. The method should draw the bullet point centered around the origin. The size of the bullet point will be multiplied by the current font size.

Definition at line 272 of file MarkdownRenderer.cs.

7.99.4.10 CodeBlockBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.CodeBlockBackgroundColour = Colour.FromRgb(240,
240, 245)  [get], [set]
```

The background colour for code blocks.

Definition at line 323 of file MarkdownRenderer.cs.

7.99.4.11 CodeFont

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFont = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Courier)  [get], [set]
```

The font family for code elements.

Definition at line 90 of file MarkdownRenderer.cs.

7.99.4.12 CodeFontBold

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontBold = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierBold)  [get], [set]
```

The font family for bold code elements.

Definition at line 95 of file MarkdownRenderer.cs.

7.99.4.13 CodeFontBoldItalic

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontBoldItalic = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierBoldOblique)  [get], [set]
```

The font family for bold italic code elements.

Definition at line 105 of file MarkdownRenderer.cs.

7.99.4.14 CodeFontItalic

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontItalic = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierOblique)  [get], [set]
```

The font family for italic code elements.

Definition at line 100 of file MarkdownRenderer.cs.

7.99.4.15 CodeInlineBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.CodeInlineBackgroundColour = Colour.FromRgb(240,
240, 240)  [get], [set]
```

The background colour for code inlines.

Definition at line 318 of file MarkdownRenderer.cs.

7.99.4.16 CodeInlineMargin

```
double VectSharp.Markdown.MarkdownRenderer.CodeInlineMargin = 0.25  [get], [set]
```

The margin at the left and right of code inlines. This value will be multiplied by the current font size.

Definition at line 186 of file MarkdownRenderer.cs.

7.99.4.17 ForegroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.ForegroundColour = Colours.Black  [get], [set]
```

The foreground colour for text elements.

Definition at line 293 of file MarkdownRenderer.cs.

7.99.4.18 HeaderFontSizeMultipliers

```
double [] VectSharp.Markdown.MarkdownRenderer.HeaderFontSizeMultipliers  [get]
```

Initial value:

```
= new double[]
        {
            28 / 12.0, 22 / 12.0, 16 / 12.0, 14 / 12.0, 13 / 12.0, 12 / 12.0
        }
```

The font size for elements at each header level. The values in this array will be multiplied by the BaseFontSize.

Definition at line 52 of file MarkdownRenderer.cs.

7.99.4.19 HeaderLineColour

```
Colour VectSharp.Markdown.MarkdownRenderer.HeaderLineColour = Colour.FromRgb(180, 180, 180)
[get], [set]
```

The colour of the line below headers.

Definition at line 303 of file MarkdownRenderer.cs.

7.99.4.20 HeaderLineThicknesses

```
double [] VectSharp.Markdown.MarkdownRenderer.HeaderLineThicknesses = new double[] { 1, 1, 0,
0, 0, 0 }  [get]
```

The thickness of the separator line after a header of each level. A value of 0 disables the line after headers of that level.

Definition at line 60 of file MarkdownRenderer.cs.

7.99.4.21 ImageMarginTolerance

```
double VectSharp.Markdown.MarkdownRenderer.ImageMarginTolerance = 25  [get], [set]
```

Images will be allowed to extend into the page bottom margin area by this amount before triggering a page break. This should be smaller than the bottom margin, otherwise images risk being cut off by the page boundary.

Definition at line 261 of file MarkdownRenderer.cs.

7.99.4.22 ImageMultiplier

```
double VectSharp.Markdown.MarkdownRenderer.ImageMultiplier = 1  [get], [set]
```

The size of images will be multiplied by this value to determine the actual size of the image on the page. For images that have a width or height attribute, this will be applied in addition to the ImageUnitMultiplier. For images without width and height, only this multiplier will be applied.

Definition at line 251 of file MarkdownRenderer.cs.

7.99.4.23 ImageSideMargin

```
double VectSharp.Markdown.MarkdownRenderer.ImageSideMargin = 10  [get], [set]
```

The margin on the right of left-aligned images and on the left of right-aligned images.

Definition at line 256 of file MarkdownRenderer.cs.

7.99.4.24 ImageUnitMultiplier

```
double VectSharp.Markdown.MarkdownRenderer.ImageUnitMultiplier = 0.60714  [get], [set]
```

The size of images (as defined in the image's width and height attributes) will be multiplied by this value to determine the actual size of the image on the page. This has no effect on images without a width or height attribute.

Definition at line 246 of file MarkdownRenderer.cs.

7.99.4.25 ImageUriResolver

```
Func<string, string, (string, bool)> VectSharp.Markdown.MarkdownRenderer.ImageUriResolver =
HTTPUtils.ResolveImageURI  [get], [set]
```

A method used to resolve (possibly remote) image uris into local file paths. The first argument of the method should be the image uri and the second argument the base uri used to resolve relative links. The method should return a tuple containing the path of the local file and a boolean value indicating whether the file has been fetched from a remote location and should be deleted after the program has finished using it.

Definition at line 226 of file MarkdownRenderer.cs.

7.99.4.26 IndentWidth

```
double VectSharp.Markdown.MarkdownRenderer.IndentWidth = 40  [get], [set]
```

The indentation width used for list items.

Definition at line 191 of file MarkdownRenderer.cs.

7.99.4.27 InsertedColour

```
Colour VectSharp.Markdown.MarkdownRenderer.InsertedColour = Colour.FromRgb(0, 158, 115)  [get],
[set]
```

The colour for text that has been styled as "inserted".

Definition at line 338 of file MarkdownRenderer.cs.

7.99.4.28 ItalicFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.ItalicFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaOblique)  [get], [set]
```

The font family for italic text.

Definition at line 80 of file MarkdownRenderer.cs.

7.99.4.29 LinkColour

```
Colour VectSharp.Markdown.MarkdownRenderer.LinkColour = Colour.FromRgb(25, 140, 191)  [get],
[set]
```

The colour for hypertext links-

Definition at line 313 of file MarkdownRenderer.cs.

7.99.4.30 LinkUriResolver

```
Func<string, string> VectSharp.Markdown.MarkdownRenderer.LinkUriResolver = a => a  [get],
[set]
```

A method used to resolve link addresses. The argument of the method should be the absolute link, and the method should return the resolved address. This can be used to "redirect" links to a different target.

Definition at line 236 of file MarkdownRenderer.cs.

7.99.4.31 Margins

```
Margins VectSharp.Markdown.MarkdownRenderer.Margins = new Margins(55, 55, 55, 55)  [get], [set]
```

The margins of the page.

Definition at line 120 of file MarkdownRenderer.cs.

7.99.4.32 MarkedColour

```
Colour VectSharp.Markdown.MarkdownRenderer.MarkedColour = Colour.FromRgb(213, 94, 0)  [get],
[set]
```

The colour for text that has been styled as "marked".

Definition at line 343 of file MarkdownRenderer.cs.

7.99.4.33 MathFontScalingFactor

```
double VectSharp.Markdown.MarkdownRenderer.MathFontScalingFactor = 0.85  [get], [set]
```

Scaling factor for the font size to use when rendering math.

Definition at line 47 of file MarkdownRenderer.cs.

7.99.4.34 PageSize

```
Size VectSharp.Markdown.MarkdownRenderer.PageSize = new Size(595, 842)  [get], [set]
```

The size of the page.

Definition at line 156 of file MarkdownRenderer.cs.

7.99.4.35 QuoteBlockBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.QuoteBlockBackgroundColour = Colour.FromRgb(240,
240, 255)  [get], [set]
```

The background colour for block quotes.

Definition at line 333 of file MarkdownRenderer.cs.

7.99.4.36 QuoteBlockBarColour

```
Colour VectSharp.Markdown.MarkdownRenderer.QuoteBlockBarColour = Colour.FromRgb(75, 152, 220)
[get], [set]
```

The colour for the bar to the left of block quotes.

Definition at line 328 of file MarkdownRenderer.cs.

7.99.4.37 QuoteBlockBarWidth

```
double VectSharp.Markdown.MarkdownRenderer.QuoteBlockBarWidth = 5  [get], [set]
```

The thickness of the bar to the left of block quotes.

Definition at line 201 of file MarkdownRenderer.cs.

7.99.4.38 QuoteBlockIndentWidth

```
double VectSharp.Markdown.MarkdownRenderer.QuoteBlockIndentWidth = 30  [get], [set]
```

The indentation width used for block quotes.

Definition at line 196 of file MarkdownRenderer.cs.

7.99.4.39 RasterImageLoader

```
Func<string, RasterImage> VectSharp.Markdown.MarkdownRenderer.RasterImageLoader = null  [get],
[set]
```

A method used to a load raster image from a local file. The argument of the method should be the path of a local image file, and the method should return a RasterImage representing that file. For example, this can be achieved using the `RasterImageFile` class from the `VectSharp.MuPDFUtils` package. If this is `null`, only SVG images will be included in the document.

Definition at line 241 of file MarkdownRenderer.cs.

7.99.4.40 RegularFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.RegularFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Helvetica)  [get], [set]
```

The font family for regular text.

Definition at line 70 of file MarkdownRenderer.cs.

7.99.4.41 SpaceAfterHeading

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterHeading = 0.25 [get], [set]
```

The space after each heading. This value will be multiplied by the font size of the heading.

Definition at line 181 of file MarkdownRenderer.cs.

7.99.4.42 SpaceAfterLine

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterLine = 0.25 [get], [set]
```

The space after each line of text. This value will be multiplied by the BaseFontSize.

Definition at line 171 of file MarkdownRenderer.cs.

7.99.4.43 SpaceAfterParagraph

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterParagraph = 0.75 [get], [set]
```

The space after each text paragraph. This value will be multiplied by the BaseFontSize.

Definition at line 166 of file MarkdownRenderer.cs.

7.99.4.44 SpaceBeforeHeading

```
double VectSharp.Markdown.MarkdownRenderer.SpaceBeforeHeading = 0.25 [get], [set]
```

The space before each heading. This value will be multiplied by the font size of the heading.

Definition at line 176 of file MarkdownRenderer.cs.

7.99.4.45 SpaceBeforeParagaph

```
double VectSharp.Markdown.MarkdownRenderer.SpaceBeforeParagaph = 0 [get], [set]
```

The space before each text paragraph. This value will be multiplied by the BaseFontSize.

Definition at line 161 of file MarkdownRenderer.cs.

7.99.4.46 SubscriptShift

```
double VectSharp.Markdown.MarkdownRenderer.SubscriptShift = 0.14  [get], [set]
```

The downwards shift in the baseline for subscript elements. This value will be multiplied by the current font size.

Definition at line 216 of file MarkdownRenderer.cs.

7.99.4.47 SubSuperscriptFontSize

```
double VectSharp.Markdown.MarkdownRenderer.SubSuperscriptFontSize = 0.7  [get], [set]
```

The font size for subscripts and superscripts. This value will be multiplied by the current font size.

Definition at line 206 of file MarkdownRenderer.cs.

7.99.4.48 SuperscriptShift

```
double VectSharp.Markdown.MarkdownRenderer.SuperscriptShift = 0.33  [get], [set]
```

The upwards shift in the baseline for superscript elements. This value will be multiplied by the current font size.

Definition at line 211 of file MarkdownRenderer.cs.

7.99.4.49 SyntaxHighlighter

```
Func<string, string, List<List<FormattedString> > > VectSharp.Markdown.MarkdownRenderer.↩
SyntaxHighlighter = VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlightedLines  [get],
[set]
```

A method used for syntax highlighting. The first argument should be the source code to highlight, while the second parameter is the name of the language to use for the highlight. The method should return a list of lists of FormattedStrings, with each list of FormattedStrings representing a line. For each code block, if the method returns `null`, no syntax highlighting is used.

Definition at line 266 of file MarkdownRenderer.cs.

7.99.4.50 TableCellMargins

```
Margins VectSharp.Markdown.MarkdownRenderer.TableCellMargins = new Margins(5, 0, 5, 0)  [get],
[set]
```

The margins for table cells.

Definition at line 125 of file MarkdownRenderer.cs.

7.99.4.51 TableHeaderRowSeparatorColour

```
Colour VectSharp.Markdown.MarkdownRenderer.TableHeaderRowSeparatorColour = Colours.Black  [get], [set]
```

The colour for the line separating the table header row from normal rows.

Definition at line 348 of file MarkdownRenderer.cs.

7.99.4.52 TableHeaderRowSeparatorThickness

```
double VectSharp.Markdown.MarkdownRenderer.TableHeaderRowSeparatorThickness = 2  [get], [set]
```

The thickness of the line separating the table header row from normal rows.

Definition at line 358 of file MarkdownRenderer.cs.

7.99.4.53 TableHeaderSeparatorThickness

```
double VectSharp.Markdown.MarkdownRenderer.TableHeaderSeparatorThickness = 1  [get], [set]
```

The thickness of lines separating table rows from each other.

Definition at line 363 of file MarkdownRenderer.cs.

7.99.4.54 TableRowSeparatorColour

```
Colour VectSharp.Markdown.MarkdownRenderer.TableRowSeparatorColour = Colour.FromRgb(180, 180, 180)  [get], [set]
```

The colour for lines separating table rows from each other.

Definition at line 353 of file MarkdownRenderer.cs.

7.99.4.55 TableVAlign

```
VerticalAlignment VectSharp.Markdown.MarkdownRenderer.TableVAlign = VerticalAlignment.Middle  [get], [set]
```

The vertical alignment of table cells.

Definition at line 151 of file MarkdownRenderer.cs.

7.99.4.56 TaskListCheckedBullet

```
Graphics VectSharp.Markdown.MarkdownRenderer.TaskListCheckedBullet  [get], [set]
```

Initial value:

```
= new Func<Graphics>(() =>
        {
            Graphics tbr = new Graphics();
            GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3, -0.4).Arc(-0.3,
      -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0, Math.PI /
      2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7, -0.2, 0.2,
      Math.PI, 3 * Math.PI / 2).Close();
            tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
            tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
            GraphicsPath tickpath = new GraphicsPath().MoveTo(-0.75, -0.1).LineTo(-0.5, 0.15).LineTo(-0.1,
      -0.4);
            tbr.StrokePath(new GraphicsPath().MoveTo(-0.5, 0.15).LineTo(-0.1, -0.4), Colour.FromRgb(240,
      246, 249), 0.3, LineCaps.Round);
            tbr.StrokePath(tickpath, Colour.FromRgb(0, 158, 115), 0.2, LineCaps.Round);
            return tbr;
        })()
```

The bullet used for checked task list items.

Definition at line 383 of file MarkdownRenderer.cs.

7.99.4.57 TaskListUncheckedBullet

```
Graphics VectSharp.Markdown.MarkdownRenderer.TaskListUncheckedBullet  [get], [set]
```

Initial value:

```
= new Func<Graphics>(() =>
        {
            Graphics tbr = new Graphics();
            GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3, -0.4).Arc(-0.3,
      -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0, Math.PI /
      2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7, -0.2, 0.2,
      Math.PI, 3 * Math.PI / 2).Close();
            tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
            tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
            return tbr;
        })()
```

The bullet used for unchecked task list items.

Definition at line 368 of file MarkdownRenderer.cs.

7.99.4.58 ThematicBreakLineColour

```
Colour VectSharp.Markdown.MarkdownRenderer.ThematicBreakLineColour = Colour.FromRgb(180, 180,
200)  [get], [set]
```

The colour for thematic break lines.

Definition at line 308 of file MarkdownRenderer.cs.

7.99.4.59 ThematicBreakThickness

```
double VectSharp.Markdown.MarkdownRenderer.ThematicBreakThickness = 2  [get], [set]
```

The thickness of thematic break lines.

Definition at line 65 of file MarkdownRenderer.cs.

7.99.4.60 UnderlineThickness

```
double VectSharp.Markdown.MarkdownRenderer.UnderlineThickness = 0.075  [get], [set]
```

The thickness of underlines. This value will be multiplied by the font size of the element being underlined.

Definition at line 110 of file MarkdownRenderer.cs.

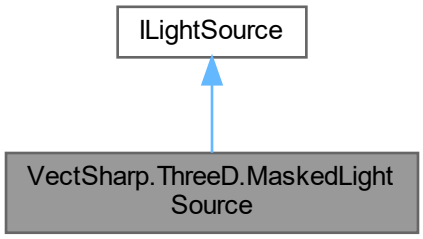
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.cs

7.100 VectSharp.ThreeD.MaskedLightSource Class Reference

Represents a point light source with a stencil in front of it.

Inheritance diagram for VectSharp.ThreeD.MaskedLightSource:



Public Member Functions

- MaskedLightSource (double intensity, Point3D position, NormalizedVector3D direction, double distance, GraphicsPath mask, double maskOrientation, double triangulationResolution)

  Creates a new MaskedLightSource by triangulating the specified GraphicsPath.
- MaskedLightSource (double intensity, Point3D position, NormalizedVector3D direction, double distance, IEnumerable< GraphicsPath > triangulatedMask, double maskOrientation)

  Creates a new MaskedLightSource using the specified triangulatedMask .
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get]`

  The position of the light source.
- Point3D Origin `[get]`

  The projection of the Position on the mask plane along the light's Direction.
- NormalizedVector3D Direction `[get]`

  The direction of the light.
- double Distance `[get]`

  The distance between the light source and the mask plane.
- double Intensity `[get, set]`

  The base intensity of the light.
- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.
- double AngleAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates away from the light's axis. Set to 0 to disable angular attenuation.

7.100.1 Detailed Description

Represents a point light source with a stencil in front of it.

Definition at line 385 of file Lights.cs.

7.100.2 Constructor & Destructor Documentation

7.100.2.1 MaskedLightSource() [1/2]

```
VectSharp.ThreeD.MaskedLightSource.MaskedLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double distance,
            GraphicsPath mask,
            double maskOrientation,
            double triangulationResolution )
```

Creates a new MaskedLightSource by triangulating the specified GraphicsPath.

Parameters

intensity	The base intensity of the light.
position	The position of the light source.
direction	The direction of the light.
distance	The distance between the light source and the mask plane.
mask	A GraphicsPath representing the transparent part of the mask.
maskOrientation	An angle in radians determining the orientation of the 2D mask in the mask plane.
triangulationResolution	The resolution to use to triangulate the *mask* .

Definition at line 437 of file Lights.cs.

7.100.2.2 MaskedLightSource() [2/2]

```
VectSharp.ThreeD.MaskedLightSource.MaskedLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double distance,
            IEnumerable< GraphicsPath > triangulatedMask,
            double maskOrientation )
```

Creates a new MaskedLightSource using the specified *triangulatedMask* .

Parameters

intensity	The base intensity of the light.
position	The position of the light source.
direction	The direction of the light.
distance	The distance between the light source and the mask plane.
triangulatedMask	A collection of GraphicsPaths representing the transparent part of the mask. Each GraphicsPath should represent a single triangle.
maskOrientation	An angle in radians determining the orientation of the 2D mask in the mask plane.

Definition at line 451 of file Lights.cs.

7.100.3 Member Function Documentation

7.100.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.MaskedLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 483 of file Lights.cs.

7.100.3.2 GetObstruction()

```
double VectSharp.ThreeD.MaskedLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 556 of file Lights.cs.

7.100.4 Property Documentation

7.100.4.1 AngleAttenuationExponent

```
double VectSharp.ThreeD.MaskedLightSource.AngleAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates away from the light's axis. Set to 0 to disable angular attenuation.

Definition at line 425 of file Lights.cs.

7.100.4.2 CastsShadow

```
bool VectSharp.ThreeD.MaskedLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 388 of file Lights.cs.

7.100.4.3 Direction

```
NormalizedVector3D VectSharp.ThreeD.MaskedLightSource.Direction  [get]
```

The direction of the light.

Definition at line 403 of file Lights.cs.

7.100.4.4 Distance

```
double VectSharp.ThreeD.MaskedLightSource.Distance  [get]
```

The distance between the light source and the mask plane.

Definition at line 408 of file Lights.cs.

7.100.4.5 DistanceAttenuationExponent

```
double VectSharp.ThreeD.MaskedLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 420 of file Lights.cs.

7.100.4.6 Intensity

```
double VectSharp.ThreeD.MaskedLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 415 of file Lights.cs.

7.100.4.7 Origin

```
Point3D VectSharp.ThreeD.MaskedLightSource.Origin  [get]
```

The projection of the Position on the mask plane along the light's Direction.

Definition at line 398 of file Lights.cs.

7.100.4.8 Position

```
Point3D VectSharp.ThreeD.MaskedLightSource.Position  [get]
```

The position of the light source.

Definition at line 393 of file Lights.cs.

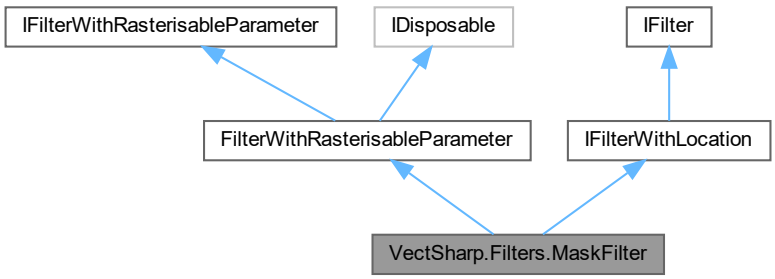
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.101 VectSharp.Filters.MaskFilter Class Reference

Represents a filter that uses the luminance of an image to mask another image.

Inheritance diagram for VectSharp.Filters.MaskFilter:



Public Member Functions

- MaskFilter (Graphics mask)

  Creates a new MaskFilter with the specified mask image.
- RasterImage Filter (RasterImage image, Rectangle bounds, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
bounds	*The region on the graphics surface where the image will be drawn.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- Graphics Mask `[get]`

  The image that is used to mask the input image.

7.101.1 Detailed Description

Represents a filter that uses the luminance of an image to mask another image.

Definition at line 26 of file MaskFilter.cs.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 MaskFilter()

```
VectSharp.Filters.MaskFilter.MaskFilter (
            Graphics mask )
```

Creates a new MaskFilter with the specified mask image.

Parameters

mask	The image that is used to mask the input image.

Definition at line 42 of file MaskFilter.cs.

7.101.3 Member Function Documentation

7.101.3.1 Filter()

```
RasterImage VectSharp.Filters.MaskFilter.Filter (
            RasterImage image,
            Rectangle bounds,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
bounds	The region on the graphics surface where the image will be drawn.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.IFilterWithLocation.

Definition at line 45 of file MaskFilter.cs.

7.101.4 Property Documentation

7.101.4.1 BottomRightMargin

```
Point VectSharp.Filters.MaskFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 31 of file MaskFilter.cs.

7.101.4.2 Mask

```
Graphics VectSharp.Filters.MaskFilter.Mask  [get]
```

The image that is used to mask the input image.

Definition at line 36 of file MaskFilter.cs.

7.101.4.3 TopLeftMargin

```
Point VectSharp.Filters.MaskFilter.TopLeftMargin [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 29 of file MaskFilter.cs.

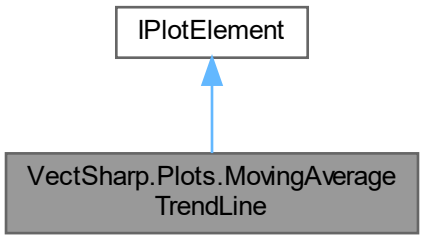
The documentation for this class was generated from the following file:

- VectSharp/Filters/MaskFilter.cs

7.102 VectSharp.Plots.MovingAverageTrendLine Class Reference

A plot element that draws a moving average trendline.

Inheritance diagram for VectSharp.Plots.MovingAverageTrendLine:



Public Member Functions

- MovingAverageTrendLine (IReadOnlyList< IReadOnlyList< double > > data, int period, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new MovingAverageTrendLine instance from the specified data.
- MovingAverageTrendLine (IReadOnlyList<(double, double)> data, int period, IContinuousCoordinateSystem coordinateSystem)

  Create a new MovingAverageTrendLine instance from the specified data.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< IReadOnlyList< double > > Data `[get, set]`

  The data used to compute the moving average. These must be sorted in a sensible way.
- Func< int, double[], double > Weight = (int i, double[] d) => 1 `[get, set]`

  The weight function. This should accept two parameters: an `int` representing the difference in index between the point being weighted and the "focus point", and a `double[]` representing the difference in coordinates between the two points. It should return a `double` representing the weight (it does not need to be normalised).
- int Period `[get, set]`

  The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.102.1 Detailed Description

A plot element that draws a moving average trendline.

Definition at line 1074 of file Trendlines.cs.

7.102.2 Constructor & Destructor Documentation

7.102.2.1 MovingAverageTrendLine() [1/2]

```
VectSharp.Plots.MovingAverageTrendLine.MovingAverageTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            int period,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new MovingAverageTrendLine instance from the specified data.

Parameters

data	The data used to compute the moving average. These must be sorted in a sensible way.
period	The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Exceptions

ArgumentOutOfRangeException	Thrown if the *period* is < 0.

Definition at line 1117 of file Trendlines.cs.

7.102.2.2 MovingAverageTrendLine() [2/2]

```
VectSharp.Plots.MovingAverageTrendLine.MovingAverageTrendLine (
            IReadOnlyList<(double, double)> data,
            int period,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new MovingAverageTrendLine instance from the specified data.

Parameters

data	The data used to compute the moving average. These must be sorted in a sensible way.
period	The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Exceptions

ArgumentOutOfRangeException	Thrown if the *period* is < 0.

Definition at line 1136 of file Trendlines.cs.

7.102.3 Member Function Documentation

7.102.3.1 Plot()

```
void VectSharp.Plots.MovingAverageTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 1139 of file Trendlines.cs.

7.102.4 Property Documentation

7.102.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.MovingAverageTrendLine.Coordinate↩
System [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 1107 of file Trendlines.cs.

7.102.4.2 Data

```
IReadOnlyList<IReadOnlyList<double> > VectSharp.Plots.MovingAverageTrendLine.Data [get],
[set]
```

The data used to compute the moving average. These must be sorted in a sensible way.

Definition at line 1079 of file Trendlines.cs.

7.102.4.3 Period

```
int VectSharp.Plots.MovingAverageTrendLine.Period [get], [set]
```

The number of points that are averaged to obtain the value for each point. This must be ≥ 0.

Definition at line 1092 of file Trendlines.cs.

7.102.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.MovingAverageTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) } [get], [set]
```

Presentation attributes for the trendline.

Definition at line 1097 of file Trendlines.cs.

7.102.4.5 Tag

```
string VectSharp.Plots.MovingAverageTrendLine.Tag [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 1102 of file Trendlines.cs.

7.102.4.6 Weight

```
Func<int, double[], double> VectSharp.Plots.MovingAverageTrendLine.Weight = (int i, double[]
d) => 1  [get], [set]
```

The weight function. This should accept two parameters: an `int` representing the difference in index between the point being weighted and the "focus point", and a `double`[] representing the difference in coordinates between the two points. It should return a `double` representing the weight (it does not need to be normalised).

Definition at line 1087 of file Trendlines.cs.

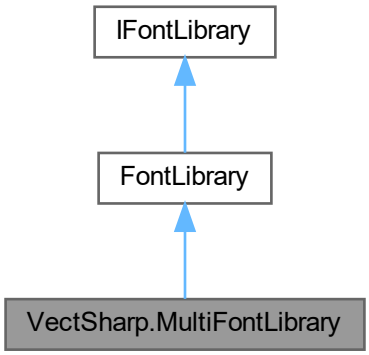
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.103 VectSharp.MultiFontLibrary Class Reference

A font library that tries to resolve fonts using other font libraries.

Inheritance diagram for VectSharp.MultiFontLibrary:



Public Member Functions

- MultiFontLibrary (params FontLibrary[ ] libraries)

  Creates a new MultiFontLibrary resolving fonts using the specified libraries .
- MultiFontLibrary (IEnumerable< FontLibrary > libraries)

  Creates a new MultiFontLibrary resolving fonts using the specified libraries .
- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.103.1 Detailed Description

A font library that tries to resolve fonts using other font libraries.

Definition at line 654 of file FontLibrary.cs.

7.103.2 Constructor & Destructor Documentation

7.103.2.1 MultiFontLibrary() [1/2]

```
VectSharp.MultiFontLibrary.MultiFontLibrary (
            params FontLibrary[] libraries )
```

Creates a new MultiFontLibrary resolving fonts using the specified *libraries* .

Parameters

libraries	The font libraries that will be used, in order, to resolve the font families.

Exceptions

ArgumentException	Thrown if the *libraries* do not contain any element.

Definition at line 664 of file FontLibrary.cs.

7.103.2.2 MultiFontLibrary() [2/2]

```
VectSharp.MultiFontLibrary.MultiFontLibrary (
            IEnumerable< FontLibrary > libraries )
```

Creates a new MultiFontLibrary resolving fonts using the specified *libraries* .

Parameters

libraries	The font libraries that will be used, in order, to resolve the font families.

Exceptions

ArgumentException	Thrown if the *libraries* do not contain any element.

Definition at line 672 of file FontLibrary.cs.

7.103.3 Member Function Documentation

7.103.3.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.MultiFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 699 of file FontLibrary.cs.

7.103.3.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.MultiFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 683 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.104 VectSharp.Fonts.Nimbus Class Reference

Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

Properties

- static IFontLibrary Library `[get]`

  The font library.

7.104.1 Detailed Description

Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

Definition at line 21 of file Nimbus.cs.

7.104.2 Property Documentation

7.104.2.1 Library

```
IFontLibrary VectSharp.Fonts.Nimbus.Library  [static], [get]
```

The font library.

Definition at line 26 of file Nimbus.cs.

The documentation for this class was generated from the following file:

- VectSharp.Fonts.Nimbus/Nimbus.cs

7.105 VectSharp.ThreeD.ObjectFactory Class Reference

A static class containing methods to create complex 3D objects.

Static Public Member Functions

- static List< Element3D > CreateCube (Point3D center, double size, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a cube.
- static List< Element3D > CreateCuboid (Point3D center, double sizeX, double sizeY, double sizeZ, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a cuboid.
- static List< Element3D > CreateRectangle (Point3D point1, Point3D point2, Point3D point3, Point3D point4, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a quadrilater. All the vertices need not be coplanar.
- static List< Element3D > CreateRectangle (Point3D point1, Point3D point2, Point3D point3, Point3D point4, NormalizedVector3D point1Normal, NormalizedVector3D point2Normal, NormalizedVector3D point3Normal, NormalizedVector3D point4Normal, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a quadrilater, specifying the vertex normals at the four vertices. All the vertices need not be coplanar.
- static List< Element3D > CreateSphere (Point3D center, double radius, int steps, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a sphere.
- static List< Element3D > CreateTetrahedron (Point3D center, double radius, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a tetrahedron inscribed in a sphere.
- static List< Element3D > CreatePolygon (GraphicsPath polygon2D, double triangulationResolution, Point3D origin, NormalizedVector3D xAxis, NormalizedVector3D yAxis, bool reverseTriangles, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a flat polygon.
- static List< Element3D > CreatePrism (GraphicsPath polygonBase2D, double triangulationResolution, Point3D bottomOrigin, Point3D topOrigin, NormalizedVector3D baseXAxis, NormalizedVector3D baseYAxis, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a prism with the specified base.
- static List< Element3D > CreateWireframe (IEnumerable< Element3D > object3D, Colour colour, double thickness=1, LineCaps lineCap=LineCaps.Butt, LineDash? lineDash=null, string tag=null, int zIndex=0)

  Creates a wireframe from a collection of Element3Ds.
- static List< Element3D > CreatePoints (IEnumerable< Element3D > object3D, Colour colour, double diameter=1, string tag=null, int zIndex=0)

  Obtains a list of Point3DElement corresponding to the vertices of a list of Element3Ds.

7.105.1 Detailed Description

A static class containing methods to create complex 3D objects.

Definition at line 28 of file ObjectFactory.cs.

7.105.2 Member Function Documentation

7.105.2.1 CreateCube()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateCube (
            Point3D center,
            double size,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a cube.

Parameters

center	The centre of the cube.
size	The length of each side of the cube.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the cube.

Definition at line 39 of file ObjectFactory.cs.

7.105.2.2 CreateCuboid()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateCuboid (
            Point3D center,
            double sizeX,
            double sizeY,
            double sizeZ,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a cuboid.

Parameters

center	The centre of the cube.
sizeX	The length of the sides of the cube parallel to the x axis.
sizeY	The length of the sides of the cube parallel to the y axis.
sizeZ	The length of the sides of the cube parallel to the z axis.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the cuboid.

Definition at line 55 of file ObjectFactory.cs.

7.105.2.3 CreatePoints()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePoints (
            IEnumerable< Element3D > object3D,
            Colour colour,
            double diameter = 1,
            string tag = null,
            int zIndex = 0 )  [static]
```

Obtains a list of Point3DElement corresponding to the vertices of a list of Element3Ds.

Parameters

object3D	The collection of Element3Ds. Point3DElements are ignored.
colour	The colour of the Point3DElements returned by this method.
diameter	The diameter of the Point3DElements returned by this method.
tag	A tag that will be applied to the Point3DElements returned by this method.
zIndex	A z-index that will be applied to the Point3DElements returned by this method.

Returns

A list of Point3DElements corresponding to the vertices of the Element3Ds.

Definition at line 412 of file ObjectFactory.cs.

7.105.2.4 CreatePolygon()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePolygon (
            GraphicsPath polygon2D,
            double triangulationResolution,
            Point3D origin,
            NormalizedVector3D xAxis,
            NormalizedVector3D yAxis,
            bool reverseTriangles,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a flat polygon.

Parameters

polygon2D	A 2D GraphicsPath representing the polygon.
triangulationResolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
origin	A Point3D that will correspond to the origin of the 2D reference system.
xAxis	A NormalizedVector3D that will correspond to the x axis of the 2D reference system. This will be orthonormalised to the *yAxis* .
yAxis	A NormalizedVector3D that will correspond to the y axis of the 2D reference system.
reverseTriangles	Indicates whether the order of the points (and thus the normals) of all the triangles returned by this method should be reversed.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the polygon.

Definition at line 273 of file ObjectFactory.cs.

7.105.2.5 CreatePrism()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePrism (
            GraphicsPath polygonBase2D,
            double triangulationResolution,
            Point3D bottomOrigin,
            Point3D topOrigin,
            NormalizedVector3D baseXAxis,
            NormalizedVector3D baseYAxis,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a prism with the specified base.

Parameters

polygonBase2D	A 2D GraphicsPath representing the base of the prism.
triangulationResolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
bottomOrigin	A Point3D that will correspond to the origin of the 2D reference system of the bottom base.
topOrigin	A Point3D that will correspond to the origin of the 2D reference system of the top base.
baseXAxis	A NormalizedVector3D that will correspond to the x axis of the 2D reference system of the bases. This will be orthonormalised to the *baseYAxis* .
baseYAxis	A NormalizedVector3D that will correspond to the y axis of the 2D reference system of the bases.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the prism.

Definition at line 314 of file ObjectFactory.cs.

7.105.2.6 CreateRectangle() [1/2]

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateRectangle (
            Point3D point1,
            Point3D point2,
            Point3D point3,
            Point3D point4,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a quadrilater. All the vertices need not be coplanar.

Parameters

point1	The first vertex of the quadrilater.
point2	The second vertex of the quadrilater.
point3	The third vertex of the quadrilater.
point4	The fourth vertex of the quadrilater.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list containing two Triangle3DElements representing the quadrilater.

Definition at line 93 of file ObjectFactory.cs.

7.105.2.7 CreateRectangle() [2/2]

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateRectangle (
            Point3D point1,
            Point3D point2,
            Point3D point3,
            Point3D point4,
            NormalizedVector3D point1Normal,
            NormalizedVector3D point2Normal,
            NormalizedVector3D point3Normal,
            NormalizedVector3D point4Normal,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a quadrilater, specifying the vertex normals at the four vertices. All the vertices need not be coplanar.

Parameters

point1	The first vertex of the quadrilater.
point2	The second vertex of the quadrilater.
point3	The third vertex of the quadrilater.
point4	The fourth vertex of the quadrilater.
point1Normal	The vertex normal at the first vertex of the quadrilater.
point2Normal	The vertex normal at the second vertex of the quadrilater.
point3Normal	The vertex normal at the third vertex of the quadrilater.
point4Normal	The vertex normal at the fourth vertex of the quadrilater.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list containing two Triangle3DElements representing the quadrilater.

Definition at line 123 of file ObjectFactory.cs.

7.105.2.8 CreateSphere()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateSphere (
            Point3D center,
            double radius,
            int steps,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a sphere.

Parameters

center	The centre of the sphere.
radius	The radius of the sphere.
steps	The number of meridians and parallels to use when generating the sphere.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the sphere.

Definition at line 148 of file ObjectFactory.cs.

7.105.2.9 CreateTetrahedron()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateTetrahedron (
            Point3D center,
            double radius,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a tetrahedron inscribed in a sphere.

Parameters

center	The centre of the tetrahedron.
radius	The radius of the sphere in which the tetrahedron is inscribed.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the sphere.

Definition at line 238 of file ObjectFactory.cs.

7.105.2.10 CreateWireframe()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateWireframe (
            IEnumerable< Element3D > object3D,
            Colour colour,
            double thickness = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineDash?  lineDash = null,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a wireframe from a collection of Element3Ds.

Parameters

object3D	The collection of Element3Ds. Line3DElements and Point3DElements are ignored.
colour	The colour of the Line3DElements returned by this method.
thickness	The thickness of the Line3DElements returned by this method.
lineCap	The line cap of the Line3DElements returned by this method.
lineDash	The line dash of the Line3DElements returned by this method.
tag	A tag that will be applied to the Line3DElements returned by this method.
zIndex	A z-index that will be applied to the Line3DElements returned by this method.

Returns

A list of Line3DElements that constitute the wireframe.

Definition at line 370 of file ObjectFactory.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/ObjectFactory.cs

7.106 VectSharp.Page Class Reference

Represents a Graphics object with a width and height.

Public Member Functions

- Page (double width, double height)
  Create a new page.
- void Crop (Rectangle region, bool removeClippedGraphics=false, string tag=null)
  Translate and resize the Page so that it displays the specified region .
- void Crop (Point topLeft, Size size, bool removeClippedGraphics=false, string tag=null)
  Translate and resize the Page so that it displays the rectangle defined by topLeft and size .
- void Crop (string tag=null)
  Translate and resize the Page so that it displays the rectangle corresponding to the bounding box of its contents.

Properties

- double Width `[get, set]`
  Width of the page.
- double Height `[get, set]`
  Height of the page.
- Graphics Graphics `[get, set]`
  Graphics surface of the page.
- Colour Background = Colour.FromRgba(255, 255, 255, 0) `[get, set]`
  Background colour of the page.

7.106.1 Detailed Description

Represents a Graphics object with a width and height.

Definition at line 47 of file Document.cs.

7.106.2 Constructor & Destructor Documentation

7.106.2.1 Page()

```
VectSharp.Page.Page (
            double width,
            double height )
```

Create a new page.

Parameters

width	The width of the page.
height	The height of the page.

Definition at line 74 of file Document.cs.

7.106.3 Member Function Documentation

7.106.3.1 Crop() [1/3]

```
void VectSharp.Page.Crop (
            Point topLeft,
            Size size,
            bool removeClippedGraphics = false,
            string tag = null )
```

Translate and resize the Page so that it displays the rectangle defined by *topLeft* and *size* .

Parameters

topLeft	The top left corner of the area to include in the page.
size	The size of the area to include in the page.
removeClippedGraphics	If this is `true`, graphics actions that fall outside of the specified region are completely removed from the plot, otherwise they are just hidden.
tag	A tag to identify the transform.

Definition at line 101 of file Document.cs.

7.106.3.2 Crop() [2/3]

```
void VectSharp.Page.Crop (
            Rectangle region,
            bool removeClippedGraphics = false,
            string tag = null )
```

Translate and resize the Page so that it displays the specified *region* .

Parameters

region	The area to include in the page.
removeClippedGraphics	If this is `true`, graphics actions that fall outside of the specified region are completely removed from the plot, otherwise they are just hidden.
tag	A tag to identify the transform.

Definition at line 89 of file Document.cs.

7.106.3.3 Crop() [3/3]

```
void VectSharp.Page.Crop (
            string tag = null )
```

Translate and resize the Page so that it displays the rectangle corresponding to the bounding box of its contents.

Parameters

tag	A tag to identify the transform.

Definition at line 129 of file Document.cs.

7.106.4 Property Documentation

7.106.4.1 Background

```
Colour VectSharp.Page.Background = Colour.FromRgba(255, 255, 255, 0)  [get], [set]
```

Background colour of the page.

Definition at line 67 of file Document.cs.

7.106.4.2 Graphics

```
Graphics VectSharp.Page.Graphics  [get], [set]
```

Graphics surface of the page.

Definition at line 62 of file Document.cs.

7.106.4.3 Height

```
double VectSharp.Page.Height  [get], [set]
```

Height of the page.

Definition at line 57 of file Document.cs.

7.106.4.4 Width

```
double VectSharp.Page.Width  [get], [set]
```

Width of the page.

Definition at line 52 of file Document.cs.

The documentation for this class was generated from the following file:

- VectSharp/Document.cs

7.107 VectSharp.TrueTypeFile.PairKerning Class Reference

Contains information describing how the position of two glyphs in a kerning pair should be altered.

Properties

- Point Glyph1Placement [get]

  This vector contains the displacement that should be applied to the first glyph of the pair.
- Point Glyph1Advance [get]

  This vector describes how the advance width of the first glyph should be altered.
- Point Glyph2Placement [get]

  This vector contains the displacement that should be applied to the second glyph of the pair.
- Point Glyph2Advance [get]

  This vector describes how the advance width of the second glyph should be altered.

7.107.1 Detailed Description

Contains information describing how the position of two glyphs in a kerning pair should be altered.

Definition at line 4440 of file TrueType.cs.

7.107.2 Property Documentation

7.107.2.1 Glyph1Advance

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph1Advance  [get]
```

This vector describes how the advance width of the first glyph should be altered.

Definition at line 4450 of file TrueType.cs.

7.107.2.2 Glyph1Placement

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph1Placement [get]
```

This vector contains the displacement that should be applied to the first glyph of the pair.

Definition at line 4445 of file TrueType.cs.

7.107.2.3 Glyph2Advance

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph2Advance [get]
```

This vector describes how the advance width of the second glyph should be altered.

Definition at line 4460 of file TrueType.cs.

7.107.2.4 Glyph2Placement

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph2Placement [get]
```

This vector contains the displacement that should be applied to the second glyph of the pair.

Definition at line 4455 of file TrueType.cs.

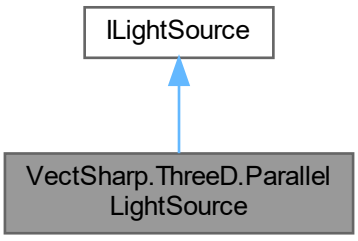
The documentation for this class was generated from the following file:

- VectSharp/TrueType.cs

7.108 VectSharp.ThreeD.ParallelLightSource Class Reference

Represents a parallel light source.

Inheritance diagram for VectSharp.ThreeD.ParallelLightSource:



Public Member Functions

- ParallelLightSource (double intensity, NormalizedVector3D direction)

  Creates a new ParallelLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- double Intensity `[get, set]`

  The intensity of the light.
- NormalizedVector3D Direction `[get]`

  The direction along which the light travels.
- NormalizedVector3D ReverseDirection `[get]`

  The reverse of Direction.
- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.

7.108.1 Detailed Description

Represents a parallel light source.

Definition at line 126 of file Lights.cs.

7.108.2 Constructor & Destructor Documentation

7.108.2.1 ParallelLightSource()

```
VectSharp.ThreeD.ParallelLightSource.ParallelLightSource (
            double intensity,
            NormalizedVector3D direction )
```

Creates a new ParallelLightSource instance.

Parameters

intensity	The intensity of the light.
direction	The direction along which the light travels.

Definition at line 151 of file Lights.cs.

7.108.3 Member Function Documentation

7.108.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.ParallelLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 159 of file Lights.cs.

7.108.3.2 GetObstruction()

```
double VectSharp.ThreeD.ParallelLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 165 of file Lights.cs.

7.108.4 Property Documentation

7.108.4.1 CastsShadow

```
bool VectSharp.ThreeD.ParallelLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 144 of file Lights.cs.

7.108.4.2 Direction

```
NormalizedVector3D VectSharp.ThreeD.ParallelLightSource.Direction  [get]
```

The direction along which the light travels.

Definition at line 136 of file Lights.cs.

7.108.4.3 Intensity

```
double VectSharp.ThreeD.ParallelLightSource.Intensity  [get], [set]
```

The intensity of the light.

Definition at line 131 of file Lights.cs.

7.108.4.4 ReverseDirection

```
NormalizedVector3D VectSharp.ThreeD.ParallelLightSource.ReverseDirection  [get]
```

The reverse of Direction.

Definition at line 141 of file Lights.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.109 VectSharp.SVG.Parser Class Reference

Contains methods to read an SVG image file.

Static Public Member Functions

- static Page ParseSVGURI (string uri, bool ignored=false)

  Parses an SVG image URI.
- static Page FromString (string svgSource)

  Parses SVG source into a Page containing the image represented by the code.
- static Page FromFile (string fileName)

  Parses an SVG image file into a Page containing the image.
- static Page FromStream (Stream svgSourceStream)

  Parses an stream containing SVG source code into a Page containing the image represented by the code.

Static Public Attributes

- static Func< string, bool, Page > ParseImageURI

  A function that takes as input an image URI and a boolean value indicating whether the image should be interpolated, and returns a Page object containing the image. By default, this is equal to ParseSVGURI, i.e. it is only able to parse SVG images. If you wish to enable the parsing of other formats, you should install the "VectSharp.MuPDFUtils" NuGet package and enable the parser in your program by doing something like:

7.109.1 Detailed Description

Contains methods to read an SVG image file.

Definition at line 35 of file SVGParser.cs.

7.109.2 Member Function Documentation

7.109.2.1 FromFile()

```
static Page VectSharp.SVG.Parser.FromFile (
            string fileName )  [static]
```

Parses an SVG image file into a Page containing the image.

Parameters

fileName	The path to the SVG image file.

Returns

A Page containing the image represented by the file.

Definition at line 200 of file SVGParser.cs.

7.109.2.2 FromStream()

```
static Page VectSharp.SVG.Parser.FromStream (
            Stream svgSourceStream )  [static]
```

Parses an stream containing SVG source code into a Page containing the image represented by the code.

Parameters

svgSourceStream	The stream containing SVG source code.

Returns

A Page containing the image represented by the *svgSourceStream* .

Definition at line 210 of file SVGParser.cs.

7.109.2.3 FromString()

```
static Page VectSharp.SVG.Parser.FromString (
            string svgSource )  [static]
```

Parses SVG source into a Page containing the image represented by the code.

Parameters

svgSource	The SVG source code.

Returns

A Page containing the image represented by the *svgSource* .

Definition at line 118 of file SVGParser.cs.

7.109.2.4 ParseSVGURI()

```
static Page VectSharp.SVG.Parser.ParseSVGURI (
            string uri,
            bool ignored = false )  [static]
```

Parses an SVG image URI.

Parameters

uri	The image URI to parse.
ignored	This value is ignored and is only needed for compatibility.

Returns

A Page containing the parsed SVG image, or null.

Definition at line 56 of file SVGParser.cs.

7.109.3 Member Data Documentation

7.109.3.1 ParseImageURI

```
Func<string, bool, Page> VectSharp.SVG.Parser.ParseImageURI  [static]
```

A function that takes as input an image URI and a boolean value indicating whether the image should be interpolated, and returns a Page object containing the image. By default, this is equal to ParseSVGURI, i.e. it is only able to parse SVG images. If you wish to enable the parsing of other formats, you should install the "VectSharp.Mu↩ PDFUtils" NuGet package and enable the parser in your program by doing something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.MuPDFUtils.ImageURIParser.Parser(VectShar
```

Definition at line 48 of file SVGParser.cs.

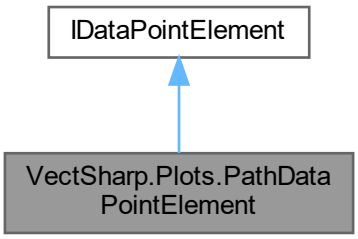
The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGParser.cs

7.110 VectSharp.Plots.PathDataPointElement Class Reference

A symbol defined by a GraphicsPath.

Inheritance diagram for VectSharp.Plots.PathDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- PathDataPointElement ()

  Create a new PathDataPointElement instance representing a circle.
- PathDataPointElement (GraphicsPath path)

  Create a new PathDataPointElement instance with the specified path .

Properties

- GraphicsPath Path = new GraphicsPath().Arc(0, 0, 1, 0, 2 * Math.PI).Close() `[get, set]`

  The GraphicsPath that constitutes the symbol (by default, a circle).

7.110.1 Detailed Description

A symbol defined by a GraphicsPath.

Definition at line 40 of file DataPoints.cs.

7.110.2 Constructor & Destructor Documentation

7.110.2.1 PathDataPointElement() [1/2]

```
VectSharp.Plots.PathDataPointElement.PathDataPointElement ( )
```

Create a new PathDataPointElement instance representing a circle.

Definition at line 79 of file DataPoints.cs.

7.110.2.2 PathDataPointElement() [2/2]

```
VectSharp.Plots.PathDataPointElement.PathDataPointElement (
            GraphicsPath path )
```

Create a new PathDataPointElement instance with the specified *path* .

Definition at line 87 of file DataPoints.cs.

7.110.3 Member Function Documentation

7.110.3.1 Plot()

```
void VectSharp.Plots.PathDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

Definition at line 48 of file DataPoints.cs.

7.110.4 Property Documentation

7.110.4.1 Path

```
GraphicsPath VectSharp.Plots.PathDataPointElement.Path = new GraphicsPath().Arc(0, 0, 1, 0, 2
* Math.PI).Close()  [get], [set]
```

The GraphicsPath that constitutes the symbol (by default, a circle).

Definition at line 45 of file DataPoints.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.111 VectSharp.PDF.PDFContextInterpreter Class Reference

Contains methods to render a Document as a PDF document.

Classes

- class FilterOption

  Determines how and whether image filters are rasterised.

Public Types

- enum TextOptions

  Defines whether the used fonts should be included in the file.

Static Public Member Functions

- static void SaveAsPDF (this Document document, string fileName, TextOptions textOption=TextOptions.↩ SubsetFonts, bool compressStreams=true, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Save the document to a PDF file.
- static void SaveAsPDF (this Document document, Stream stream, TextOptions textOption=TextOptions.↩ SubsetFonts, bool compressStreams=true, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Save the document to a PDF stream.

7.111.1 Detailed Description

Contains methods to render a Document as a PDF document.

Definition at line 831 of file PDFContext.cs.

7.111.2 Member Enumeration Documentation

7.111.2.1 TextOptions

```
enum VectSharp.PDF.PDFContextInterpreter.TextOptions
```

Defines whether the used fonts should be included in the file.

Definition at line 1126 of file PDFContext.cs.

7.111.3 Member Function Documentation

7.111.3.1 SaveAsPDF() [1/2]

```
static void VectSharp.PDF.PDFContextInterpreter.SaveAsPDF (
            this Document document,
            Stream stream,
            TextOptions textOption = TextOptions.SubsetFonts,
            bool compressStreams = true,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Save the document to a PDF stream.

Parameters

document	The Document to save.
stream	The stream to which the PDF data will be written.
textOption	Defines whether the used fonts should be included in the file.
compressStreams	Indicates whether the streams in the PDF file should be compressed.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 1210 of file PDFContext.cs.

7.111.3.2 SaveAsPDF() [2/2]

```
static void VectSharp.PDF.PDFContextInterpreter.SaveAsPDF (
            this Document document,
```

```
string fileName,
TextOptions textOption = TextOptions.SubsetFonts,
bool compressStreams = true,
Dictionary< string, string > linkDestinations = null,
FilterOption filterOption = default )  [static]
```

Save the document to a PDF file.

Parameters

document	The Document to save.
fileName	The full path to the file to save. If it exists, it will be overwritten.
textOption	Defines whether the used fonts should be included in the file.
compressStreams	Indicates whether the streams in the PDF file should be compressed.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 1115 of file PDFContext.cs.

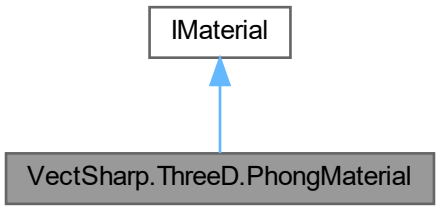
The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFContext.cs

7.112 VectSharp.ThreeD.PhongMaterial Class Reference

Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.

Inheritance diagram for VectSharp.ThreeD.PhongMaterial:



Public Member Functions

- PhongMaterial (Colour colour)

  Creates a new PhongMaterial instance.
- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

Parameters

point	*The point whose colour should be determined.*
surfaceNormal	*The normal to the surface at the specified point .*
camera	*The camera being used to render the scene.*
lights	*A list of light sources that are present in the scene.*
obstructions	*A list of values indicating how obstructed each light source is.*

Returns

The Colour of the specified point.

Properties

- Colour Colour `[get]`

  The base colour of the material.
- double AmbientReflectionCoefficient = 1 `[get, set]`

  A coefficient determining how much ambient light is reflected by the material.
- double DiffuseReflectionCoefficient = 1 `[get, set]`

  A coefficient determining how much directional light is reflected by the material.
- double SpecularReflectionCoefficient = 1 `[get, set]`

  A coefficient determining the intensity of specular highlights.
- double SpecularShininess = 1 `[get, set]`

  A coefficient determining the extent of specular highlights.

7.112.1 Detailed Description

Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.

Definition at line 74 of file Materials.cs.

7.112.2 Constructor & Destructor Documentation

7.112.2.1 PhongMaterial()

```
VectSharp.ThreeD.PhongMaterial.PhongMaterial (
            Colour colour )
```

Creates a new PhongMaterial instance.

Parameters

colour	The base colour of the material.

Definition at line 111 of file Materials.cs.

7.112.3 Member Function Documentation

7.112.3.1 GetColour()

```
Colour VectSharp.ThreeD.PhongMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implements VectSharp.ThreeD.IMaterial.

Definition at line 142 of file Materials.cs.

7.112.4 Property Documentation

7.112.4.1 AmbientReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.AmbientReflectionCoefficient = 1  [get], [set]
```

A coefficient determining how much ambient light is reflected by the material.

Definition at line 90 of file Materials.cs.

7.112.4.2 Colour

```
Colour VectSharp.ThreeD.PhongMaterial.Colour  [get]
```

The base colour of the material.

Definition at line 79 of file Materials.cs.

7.112.4.3 DiffuseReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.DiffuseReflectionCoefficient = 1  [get], [set]
```

A coefficient determining how much directional light is reflected by the material.

Definition at line 95 of file Materials.cs.

7.112.4.4 SpecularReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.SpecularReflectionCoefficient = 1  [get], [set]
```

A coefficient determining the intensity of specular highlights.

Definition at line 100 of file Materials.cs.

7.112.4.5 SpecularShininess

```
double VectSharp.ThreeD.PhongMaterial.SpecularShininess = 1  [get], [set]
```

A coefficient determining the extent of specular highlights.

Definition at line 105 of file Materials.cs.

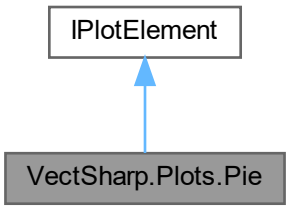
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.113 VectSharp.Plots.Pie Class Reference

A plot element that draws a pie or a doughnut.

Inheritance diagram for VectSharp.Plots.Pie:



Public Member Functions

- Pie (IReadOnlyList< double > data, IReadOnlyList< double > centre, IReadOnlyList< double > radius, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Pie instance drawing a pie chart.
- Pie (IReadOnlyList< double > data, IReadOnlyList< double > centre, IReadOnlyList< double > innerRadius, IReadOnlyList< double > outerRadius, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Pie instance drawing a doughnut chart.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Data `[get, set]`

  The data in the pie chart. The values do not need to be normalised.
- IReadOnlyList< double > Centre `[get, set]`

  The centre of the pie/doughnut, in data coordinates.
- IReadOnlyList< double > OuterRadius `[get, set]`

  The outer radius of the pie/doughnut, in data coordinates.
- IReadOnlyList< double > InnerRadius `[get, set]`

  The inner radius of the doughnut, in data coordinates. Set to [0, 0] for a pie chart.
- double StartAngle = 0 `[get, set]`

  The initial angle starting from which the pie slices are drawn.
- bool Clockwise = false `[get, set]`

  Determines whether the slices are drawn in clockwise or anti-clockwise fashion.

- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } [get, set]

  Presentation attributes for the slices. An element from this collection is used for each slice in the pie/doughnut; if there are more slices than elements in this collection, the presentation attributes are wrapped.
- string Tag [get, set]

  A tag to identify the pie/doughnut in the plot.

7.113.1 Detailed Description

A plot element that draws a pie or a doughnut.

Definition at line 27 of file Pie.cs.

7.113.2 Constructor & Destructor Documentation

7.113.2.1 Pie() [1/2]

```
VectSharp.Plots.Pie.Pie (
            IReadOnlyList< double > data,
            IReadOnlyList< double > centre,
            IReadOnlyList< double > radius,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Pie instance drawing a pie chart.

Parameters

data	The data in the pie chart. The values do not need to be normalised.
centre	The centre of the pie, in data coordinates.
radius	The radius of the pie, in data coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 84 of file Pie.cs.

7.113.2.2 Pie() [2/2]

```
VectSharp.Plots.Pie.Pie (
            IReadOnlyList< double > data,
            IReadOnlyList< double > centre,
            IReadOnlyList< double > innerRadius,
            IReadOnlyList< double > outerRadius,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Pie instance drawing a doughnut chart.

Parameters

data	The data in the doughnut chart. The values do not need to be normalised.
centre	The centre of the doughnut, in data coordinates.
innerRadius	The inner radius of the doughnut, in data coordinates.
outerRadius	The outer radius of the doughnut, in data coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 101 of file Pie.cs.

7.113.3 Member Function Documentation

7.113.3.1 Plot()

```
void VectSharp.Plots.Pie.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 111 of file Pie.cs.

7.113.4 Property Documentation

7.113.4.1 Centre

```
IReadOnlyList<double> VectSharp.Plots.Pie.Centre  [get], [set]
```

The centre of the pie/doughnut, in data coordinates.

Definition at line 37 of file Pie.cs.

7.113.4.2 Clockwise

```
bool VectSharp.Plots.Pie.Clockwise = false  [get], [set]
```

Determines whether the slices are drawn in clockwise or anti-clockwise fashion.

Definition at line 57 of file Pie.cs.

7.113.4.3 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.Pie.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 62 of file Pie.cs.

7.113.4.4 Data

```
IReadOnlyList<double> VectSharp.Plots.Pie.Data  [get], [set]
```

The data in the pie chart. The values do not need to be normalised.

Definition at line 32 of file Pie.cs.

7.113.4.5 InnerRadius

```
IReadOnlyList<double> VectSharp.Plots.Pie.InnerRadius  [get], [set]
```

The inner radius of the doughnut, in data coordinates. Set to [0, 0] for a pie chart.

Definition at line 47 of file Pie.cs.

7.113.4.6 OuterRadius

```
IReadOnlyList<double> VectSharp.Plots.Pie.OuterRadius  [get], [set]
```

The outer radius of the pie/doughnut, in data coordinates.

Definition at line 42 of file Pie.cs.

7.113.4.7 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.Pie.PresentationAttributes
= new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() }  [get],
[set]
```

Presentation attributes for the slices. An element from this collection is used for each slice in the pie/doughnut; if there are more slices than elements in this collection, the presentation attributes are wrapped.

Definition at line 70 of file Pie.cs.

7.113.4.8 StartAngle

```
double VectSharp.Plots.Pie.StartAngle = 0  [get], [set]
```

The initial angle starting from which the pie slices are drawn.

Definition at line 52 of file Pie.cs.

7.113.4.9 Tag

```
string VectSharp.Plots.Pie.Tag  [get], [set]
```

A tag to identify the pie/doughnut in the plot.

Definition at line 75 of file Pie.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Pie.cs

7.114 VectSharp.Plots.Plot Class Reference

Represents a collection of plot elements.

Public Types

- enum WhiskerType

  Describes the types of whiskers for a box plot.
- enum NormalisationMode

  Describes the kind of normalisation that is performed.

Public Member Functions

- void AddPlotElement (IPlotElement plotElement)

  Add the specified plot element to the plot.
- void AddPlotElements (IEnumerable< IPlotElement > plotElements)

  Add the specified plot elements to the plot.
- void AddPlotElements (params IPlotElement[] plotElements)

  Add the specified plot elements to the plot.
- void RemovePlotElement (IPlotElement plotElement)

  Remove the specified plot element from the plot.
- void Render (Graphics target)

  Render the plot on the specified target Graphics.
- T GetFirst< T > ()

  Get the first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type.
- IEnumerable< T > GetAll< T > ()

  Get all IPlotElements of the specified type, or all ICoordinateSystems of the specified type.
- Page Render ()

  Render the plot to a suitably cropped Page object.
- Plot ()

  Create a new empty Plot.

Properties

- ImmutableList< IPlotElement > PlotElements = ImmutableList.Create<IPlotElement>() `[get]`

  The elements contained in the plot.

7.114.1 Detailed Description

Represents a collection of plot elements.

Definition at line 24 of file Plot.Area.cs.

7.114.2 Member Enumeration Documentation

7.114.2.1 NormalisationMode

`enum VectSharp.Plots.Plot.NormalisationMode`

Describes the kind of normalisation that is performed.

Definition at line 29 of file Plot.Histogram.cs.

7.114.2.2 WhiskerType

```
enum VectSharp.Plots.Plot.WhiskerType
```

Describes the types of whiskers for a box plot.

Definition at line 30 of file Plot.BoxPlot.cs.

7.114.3 Constructor & Destructor Documentation

7.114.3.1 Plot()

```
VectSharp.Plots.Plot.Plot ( )
```

Create a new empty Plot.

Definition at line 260 of file Plot.cs.

7.114.4 Member Function Documentation

7.114.4.1 AddPlotElement()

```
void VectSharp.Plots.Plot.AddPlotElement (
            IPlotElement plotElement )
```

Add the specified plot element to the plot.

Parameters

plotElement	The plot element to add.

Definition at line 154 of file Plot.cs.

7.114.4.2 AddPlotElements() [1/2]

```
void VectSharp.Plots.Plot.AddPlotElements (
            IEnumerable< IPlotElement > plotElements )
```

Add the specified plot elements to the plot.

Parameters

plotElements	The plot elements to add.

Definition at line 163 of file Plot.cs.

7.114.4.3 AddPlotElements() [2/2]

```
void VectSharp.Plots.Plot.AddPlotElements (
            params IPlotElement[] plotElements )
```

Add the specified plot elements to the plot.

Parameters

plotElements	The plot elements to add.

Definition at line 172 of file Plot.cs.

7.114.4.4 GetAll< T >()

```
IEnumerable< T > VectSharp.Plots.Plot.GetAll< T > ( )
```

Get all IPlotElements of the specified type, or all ICoordinateSystems of the specified type.

Template Parameters

T	The type of plot element or coordinate system to get.

Returns

All IPlotElements of the specified type, or all ICoordinateSystems of the specified type.

Type Constraints

T* : *class
T* : *IPlotElement

Definition at line 225 of file Plot.cs.

7.114.4.5 GetFirst< T >()

```
T VectSharp.Plots.Plot.GetFirst< T > ( )
```

Get the first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type.

Template Parameters

T	The type of plot element or coordinate system to get.

Returns

The first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type, or `null` if none could be found.

Type Constraints

T* : *class

Definition at line 203 of file Plot.cs.

7.114.4.6 RemovePlotElement()

```
void VectSharp.Plots.Plot.RemovePlotElement (
            IPlotElement plotElement )
```

Remove the specified plot element from the plot.

Parameters

plotElement	The plot element to remove.

Definition at line 181 of file Plot.cs.

7.114.4.7 Render() [1/2]

```
Page VectSharp.Plots.Plot.Render ( )
```

Render the plot to a suitably cropped Page object.

Returns

A Page object containing the rendered plot.

Definition at line 244 of file Plot.cs.

7.114.4.8 Render() [2/2]

```
void VectSharp.Plots.Plot.Render (
            Graphics target )
```

Render the plot on the specified *target* Graphics.

Parameters

target	The Graphics on which the plot should be drawn.

Definition at line 190 of file Plot.cs.

7.114.5 Property Documentation

7.114.5.1 PlotElements

```
ImmutableList<IPlotElement> VectSharp.Plots.Plot.PlotElements = ImmutableList.Create<IPlotElement>()
[get]
```

The elements contained in the plot.

Definition at line 148 of file Plot.cs.

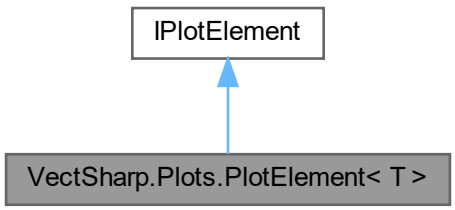
The documentation for this class was generated from the following files:

- VectSharp.Plots/Plot.Area.cs
- VectSharp.Plots/Plot.cs
- VectSharp.Plots/Plot.Histogram.cs

7.115 VectSharp.Plots.PlotElement< T > Class Template Reference

A plot element that uses an Action to draw its contents.

Inheritance diagram for VectSharp.Plots.PlotElement< T >:



Public Member Functions

- PlotElement (ICoordinateSystem< T > coordinateSystem, Action< Graphics, ICoordinateSystem< T > > plotAction)

  Create a new PlotElement< T > using the specified coordinate system and plot action.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- Action< Graphics, ICoordinateSystem< T > > PlotAction `[get, set]`

  The action that is invoked when the plot element needs to be drawn.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.115.1 Detailed Description

A plot element that uses an Action to draw its contents.

Template Parameters

T	The type of data to plot (e.g. `double[]`).

Definition at line 109 of file Plot.cs.

7.115.2 Constructor & Destructor Documentation

7.115.2.1 PlotElement()

```
VectSharp.Plots.PlotElement< T >.PlotElement (
            ICoordinateSystem< T > coordinateSystem,
            Action< Graphics, ICoordinateSystem< T > > plotAction )
```

Create a new PlotElement<T> using the specified coordinate system and plot action.

Parameters

coordinateSystem	The coordinate system used to transform the points from data space to plot space.
plotAction	The action that is invoked when the plot element needs to be drawn.

Definition at line 127 of file Plot.cs.

7.115.3 Member Function Documentation

7.115.3.1 Plot()

```
void VectSharp.Plots.PlotElement< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 134 of file Plot.cs.

7.115.4 Property Documentation

7.115.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.PlotElement< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 119 of file Plot.cs.

7.115.4.2 PlotAction

```
Action<Graphics, ICoordinateSystem<T> > VectSharp.Plots.PlotElement< T >.PlotAction  [get],
[set]
```

The action that is invoked when the plot element needs to be drawn.

Definition at line 114 of file Plot.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Plot.cs

7.116 VectSharp.Plots.PlotElementPresentationAttributes Class Reference

Determines the appearance of plot elements.

Public Member Functions

- PlotElementPresentationAttributes ()

  Create a new PlotElementPresentationAttributes with the default values.
- PlotElementPresentationAttributes (PlotElementPresentationAttributes other)

  Creates a new PlotElementPresentationAttributes copying all settings from another instance.

Properties

- Brush Stroke = Colours.Black `[get, set]`

  The stroke of the plot element.
- Brush Fill = Colours.Black `[get, set]`

  The fill of the plot element.
- double LineWidth = 1 `[get, set]`

  The thickness of lines in the plot element.
- LineDash? LineDash = null `[get, set]`

  The line dash style for the plot element.
- LineCaps LineCap = LineCaps.Square `[get, set]`

  The line cap for the plot element.
- LineJoins LineJoin = LineJoins.Miter `[get, set]`

  The line join for the plot element.
- Font Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), 12) `[get, set]`

  The font used to draw text in the plot element.

7.116.1 Detailed Description

Determines the appearance of plot elements.

Definition at line 27 of file Plot.cs.

7.116.2 Constructor & Destructor Documentation

7.116.2.1 PlotElementPresentationAttributes() [1/2]

```
VectSharp.Plots.PlotElementPresentationAttributes.PlotElementPresentationAttributes ( )
```

Create a new PlotElementPresentationAttributes with the default values.

Definition at line 67 of file Plot.cs.

7.116.2.2 PlotElementPresentationAttributes() [2/2]

```
VectSharp.Plots.PlotElementPresentationAttributes.PlotElementPresentationAttributes (
            PlotElementPresentationAttributes other )
```

Creates a new PlotElementPresentationAttributes copying all settings from another instance.

Parameters

other	The other instance from which the settings will be copied.

Definition at line 76 of file Plot.cs.

7.116.3 Property Documentation

7.116.3.1 Fill

```
Brush VectSharp.Plots.PlotElementPresentationAttributes.Fill = Colours.Black  [get], [set]
```

The fill of the plot element.

Definition at line 37 of file Plot.cs.

7.116.3.2 Font

```
Font VectSharp.Plots.PlotElementPresentationAttributes.Font = new Font(FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Helvetica), 12)  [get], [set]
```

The font used to draw text in the plot element.

Definition at line 62 of file Plot.cs.

7.116.3.3 LineCap

```
LineCaps VectSharp.Plots.PlotElementPresentationAttributes.LineCap = LineCaps.Square  [get],
[set]
```

The line cap for the plot element.

Definition at line 52 of file Plot.cs.

7.116.3.4 LineDash

```
LineDash?  VectSharp.Plots.PlotElementPresentationAttributes.LineDash = null  [get], [set]
```

The line dash style for the plot element.

Definition at line 47 of file Plot.cs.

7.116.3.5 LineJoin

```
LineJoins VectSharp.Plots.PlotElementPresentationAttributes.LineJoin = LineJoins.Miter  [get], [set]
```

The line join for the plot element.

Definition at line 57 of file Plot.cs.

7.116.3.6 LineWidth

```
double VectSharp.Plots.PlotElementPresentationAttributes.LineWidth = 1  [get], [set]
```

The thickness of lines in the plot element.

Definition at line 42 of file Plot.cs.

7.116.3.7 Stroke

```
Brush VectSharp.Plots.PlotElementPresentationAttributes.Stroke = Colours.Black  [get], [set]
```

The stroke of the plot element.

Definition at line 32 of file Plot.cs.

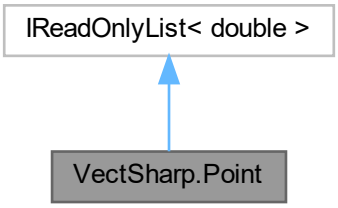
The documentation for this class was generated from the following file:

- VectSharp.Plots/Plot.cs

7.117 VectSharp.Point Struct Reference

Represents a point relative to an origin in the top-left corner.

Inheritance diagram for VectSharp.Point:



Public Member Functions

- Point (double x, double y)

  Create a new Point.
- double Modulus ()

  Computes the modulus of the vector represented by the Point.
- Point Normalize ()

  Normalises a Point.
- bool IsEqual (Point p2, double tolerance)

  Checks whether this Point is equal to another Point, up to a specified tolerance.
- IEnumerator< double > GetEnumerator ()
- IEnumerator IEnumerable. **GetEnumerator** ()

Static Public Member Functions

- static Point Min (Point p1, Point p2)

  Computes the top-left corner of the Rectangle identified by two Points.
- static Point Max (Point p1, Point p2)

  Computes the bottom-right corner of the Rectangle identified by two Points.
- static Rectangle Bounds (IEnumerable< Point > points)

  Computes the smallest Rectangle that contains all the specified points.
- static Rectangle Bounds (params Point[ ] points)

  Computes the smallest Rectangle that contains all the specified points.
- static implicit operator (double X, double Y)(Point point)

  Implicit conversion to a tuple.
- static implicit operator double[ ] (Point point)

  Implicit conversion to an array.
- static implicit operator Point ((double X, double Y) tuple)

  Implicit conversion from a tuple.
- static operator Point (double[ ] array)

  Explicit conversion from an array (will throw an exception if the array does not contain exactly two elements).
- static Point operator- (Point p1, Point p2)

  Subtract the coordinates of two points.
- static Point operator+ (Point p1, Point p2)

  Add the coordinates of two points.
- static Point operator* (double t, Point p)

  Multiply the coordinates of a point by a scalar.
- static Point operator* (Point p, double t)

  Multiply the coordinates of a point by a scalar.

Public Attributes

- double X

  Horizontal (x) coordinate, measured to the right of the origin.
- double Y

  Vertical (y) coordinate, measured to the bottom of the origin.

Properties

- int Count [get]
- double this[int index] [get]

7.117.1 Detailed Description

Represents a point relative to an origin in the top-left corner.

Definition at line 28 of file Point.cs.

7.117.2 Constructor & Destructor Documentation

7.117.2.1 Point()

```
VectSharp.Point.Point (
            double x,
            double y )
```

Create a new Point.

Parameters

x	The horizontal (x) coordinate.
y	The vertical (y) coordinate.

Definition at line 68 of file Point.cs.

7.117.3 Member Function Documentation

7.117.3.1 Bounds() [1/2]

```
static Rectangle VectSharp.Point.Bounds (
            IEnumerable< Point > points )  [static]
```

Computes the smallest Rectangle that contains all the specified points.

Parameters

points	The points whose bounds are being computed.

Returns

The smallest Rectangle that contains all the specified points.

Definition at line 131 of file Point.cs.

7.117.3.2 Bounds() [2/2]

```
static Rectangle VectSharp.Point.Bounds (
            params Point[] points )  [static]
```

Computes the smallest Rectangle that contains all the specified points.

Parameters

points	The points whose bounds are being computed.

Returns

The smallest Rectangle that contains all the specified points.

Definition at line 160 of file Point.cs.

7.117.3.3 GetEnumerator()

```
IEnumerator< double > VectSharp.Point.GetEnumerator ( )
```

Definition at line 166 of file Point.cs.

7.117.3.4 IsEqual()

```
bool VectSharp.Point.IsEqual (
            Point p2,
            double tolerance )
```

Checks whether this Point is equal to another Point, up to a specified tolerance.

Parameters

p2	The Point to compare.
tolerance	The tolerance threshold.

Returns

`true` if both coordinates of the Points are closer than *tolerance* or if their relative difference (i.e. `(a - b) / (a + b) * 2`) is smaller than *tolerance* . `false` otherwise.

Definition at line 99 of file Point.cs.

7.117.3.5 Max()

```
static Point VectSharp.Point.Max (
            Point p1,
            Point p2 )  [static]
```

Computes the bottom-right corner of the Rectangle identified by two Points.

Parameters

p1	The first point.
p2	The second point.

Returns

A Point whose X coordinate is the largest between the one of *p1* and *p2* , and likewise for the Y coordinate.

Definition at line 121 of file Point.cs.

7.117.3.6 Min()

```
static Point VectSharp.Point.Min (
            Point p1,
            Point p2 )  [static]
```

Computes the top-left corner of the Rectangle identified by two Points.

Parameters

p1	The first point.
p2	The second point.

Returns

A Point whose X coordinate is the smallest between the one of *p1* and *p2* , and likewise for the Y coordinate.

Definition at line 110 of file Point.cs.

7.117.3.7 Modulus()

```
double VectSharp.Point.Modulus ( )
```

Computes the modulus of the vector represented by the Point.

Returns

The modulus of the vector represented by the Point.

Definition at line 78 of file Point.cs.

7.117.3.8 Normalize()

```
Point VectSharp.Point.Normalize ( )
```

Normalises a Point.

Returns

The normalised Point.

Definition at line 87 of file Point.cs.

7.117.3.9 operator()

```
static implicit VectSharp.Point.operator (
            double X,
            double Y )  [static]
```

Implicit conversion to a tuple.

Parameters

point	The point to convert.

Definition at line 179 of file Point.cs.

7.117.3.10 operator double[]()

```
static implicit VectSharp.Point.operator double[] (
            Point point )  [static]
```

Implicit conversion to an array.

Parameters

point	The point to convert

Definition at line 188 of file Point.cs.

7.117.3.11 operator Point() [1/2]

```
static implicit VectSharp.Point.operator Point (
            (double X, double Y) tuple )  [static]
```

Implicit conversion from a tuple.

Parameters

tuple	The tuple to convert.

Definition at line 197 of file Point.cs.

7.117.3.12 operator Point() [2/2]

```
static VectSharp.Point.operator Point (
            double[] array )  [explicit], [static]
```

Explicit conversion from an array (will throw an exception if the array does not contain exactly two elements).

Parameters

array	The array to convert. It must contain exactly two elements.

Definition at line 206 of file Point.cs.

7.117.3.13 operator∗() [1/2]

```
static Point VectSharp.Point.operator* (
            double t,
            Point p )  [static]
```

Multiply the coordinates of a point by a scalar.

Parameters

t	The scalar.
p	The point.

Returns

A new Point whose coordinates are the product of the original points' and the scalar.

Definition at line 248 of file Point.cs.

7.117.3.14 operator∗() [2/2]

```
static Point VectSharp.Point.operator* (
            Point p,
            double t )  [static]
```

Multiply the coordinates of a point by a scalar.

Parameters

t	The scalar.
p	The point.

Returns

A new Point whose coordinates are the product of the original points' and the scalar.

Definition at line 259 of file Point.cs.

7.117.3.15 operator+()

```
static Point VectSharp.Point.operator+ (
            Point p1,
            Point p2 )  [static]
```

Add the coordinates of two points.

Parameters

p1	The first point.
p2	The second point.

Returns

A new Point whose coordinates are the sum of the coordinates of the original points.

Definition at line 237 of file Point.cs.

7.117.3.16 operator-()

```
static Point VectSharp.Point.operator- (
            Point p1,
            Point p2 )  [static]
```

Subtract the coordinates of two points.

Parameters

p1	The first point.
p2	The second point.

Returns

A new Point whose coordinates are the difference between the coordinates of the original points.

Definition at line 226 of file Point.cs.

7.117.4 Member Data Documentation

7.117.4.1 X

```
double VectSharp.Point.X
```

Horizontal (x) coordinate, measured to the right of the origin.

Definition at line 33 of file Point.cs.

7.117.4.2 Y

```
double VectSharp.Point.Y
```

Vertical (y) coordinate, measured to the bottom of the origin.

Definition at line 38 of file Point.cs.

7.117.5 Property Documentation

7.117.5.1 Count

```
int VectSharp.Point.Count  [get]
```

Definition at line 41 of file Point.cs.

7.117.5.2 this[int index]

```
double VectSharp.Point.this[int index]  [get]
```

Definition at line 44 of file Point.cs.

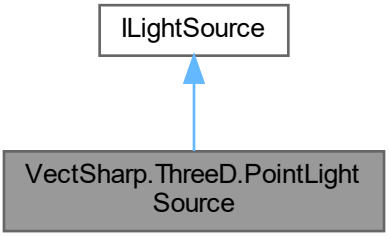
The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.118 VectSharp.ThreeD.PointLightSource Class Reference

Represents a point light source.

Inheritance diagram for VectSharp.ThreeD.PointLightSource:



Public Member Functions

- PointLightSource (double intensity, Point3D position)

  Creates a new PointLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get, set]`

  The position of the light source.
- double Intensity `[get, set]`

  The base intensity of the light.
- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

7.118.1 Detailed Description

Represents a point light source.

Definition at line 184 of file Lights.cs.

7.118.2 Constructor & Destructor Documentation

7.118.2.1 PointLightSource()

```
VectSharp.ThreeD.PointLightSource.PointLightSource (
            double intensity,
            Point3D position )
```

Creates a new PointLightSource instance.

Parameters

intensity	The intensity of the light.
position	The position of the light source.

Definition at line 209 of file Lights.cs.

7.118.3 Member Function Documentation

7.118.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.PointLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 216 of file Lights.cs.

7.118.3.2 GetObstruction()

```
double VectSharp.ThreeD.PointLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 233 of file Lights.cs.

7.118.4 Property Documentation

7.118.4.1 CastsShadow

```
bool VectSharp.ThreeD.PointLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 187 of file Lights.cs.

7.118.4.2 DistanceAttenuationExponent

```
double VectSharp.ThreeD.PointLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 202 of file Lights.cs.

7.118.4.3 Intensity

```
double VectSharp.ThreeD.PointLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 197 of file Lights.cs.

7.118.4.4 Position

```
Point3D VectSharp.ThreeD.PointLightSource.Position  [get], [set]
```

The position of the light source.

Definition at line 192 of file Lights.cs.

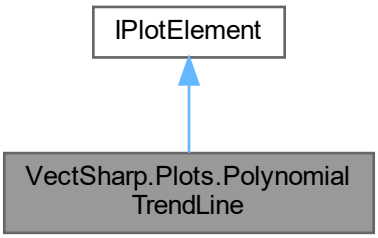
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.119 VectSharp.Plots.PolynomialTrendLine Class Reference

A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.

Inheritance diagram for VectSharp.Plots.PolynomialTrendLine:



Public Member Functions

- PolynomialTrendLine (double[] coefficients, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LinearTrendLine instance, specifying the coefficients.
- PolynomialTrendLine (IReadOnlyList< IReadOnlyList< double > > data, int order, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.
- PolynomialTrendLine (IReadOnlyList<(double, double)> data, int order, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double[] Coefficients `[get, set]`

  The coefficients (`a0 ... aN`).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

  The maximum X value for which the trendline is plotted.

- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.119.1 Detailed Description

A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.

Definition at line 685 of file Trendlines.cs.

7.119.2 Constructor & Destructor Documentation

7.119.2.1 PolynomialTrendLine() [1/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
            double[] coefficients,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LinearTrendLine instance, specifying the coefficients.

Parameters

coefficients	The coefficients (`a0 ... aN`).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 736 of file Trendlines.cs.

7.119.2.2 PolynomialTrendLine() [2/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
```

```
IReadOnlyList< IReadOnlyList< double > > data,
int order,
IContinuousCoordinateSystem coordinateSystem,
double?  fixedIntercept = null )
```

Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.

Parameters

data	The data that will be used to determine the coefficients.
order	The order of the polynomial (N). This must be ≥ 2.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (a0) is determined during the regression; otherwise, it is fixed to the specified value.

Exceptions

ArgumentOutOfRangeException	Thrown if the *order* is < 2.

Definition at line 777 of file Trendlines.cs.

7.119.2.3 PolynomialTrendLine() [3/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
            IReadOnlyList<(double, double)> data,
            int order,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.

Parameters

data	The data that will be used to determine the coefficients.
order	The order of the polynomial (N). This must be ≥ 2.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (a0) is determined during the regression; otherwise, it is fixed to the specified value.

Exceptions

ArgumentOutOfRangeException	Thrown if the *order* is < 2.

Definition at line 830 of file Trendlines.cs.

7.119.3 Member Function Documentation

7.119.3.1 Plot()

```
void VectSharp.Plots.PolynomialTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 833 of file Trendlines.cs.

7.119.4 Property Documentation

7.119.4.1 Coefficients

```
double [] VectSharp.Plots.PolynomialTrendLine.Coefficients  [get], [set]
```

The coefficients (`a0 ... aN`).

Definition at line 690 of file Trendlines.cs.

7.119.4.2 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.PolynomialTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 724 of file Trendlines.cs.

7.119.4.3 MaxX

```
double VectSharp.Plots.PolynomialTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 704 of file Trendlines.cs.

7.119.4.4 MaxY

```
double VectSharp.Plots.PolynomialTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 709 of file Trendlines.cs.

7.119.4.5 MinX

```
double VectSharp.Plots.PolynomialTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 694 of file Trendlines.cs.

7.119.4.6 MinY

```
double VectSharp.Plots.PolynomialTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 699 of file Trendlines.cs.

7.119.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.PolynomialTrendLine.PresentationAttributes =
new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 714 of file Trendlines.cs.

7.119.4.8 Tag

```
string VectSharp.Plots.PolynomialTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 719 of file Trendlines.cs.

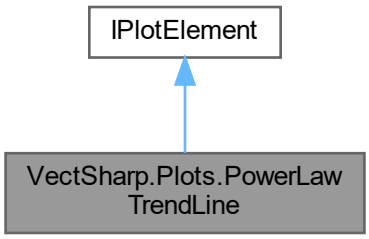
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.120 VectSharp.Plots.PowerLawTrendLine Class Reference

A plot element that draws a power law trendline with equation `y = b * x^a`.

Inheritance diagram for VectSharp.Plots.PowerLawTrendLine:



Public Member Functions

- PowerLawTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new PowerLawTrendLine instance, specifying the equation parameters.
- PowerLawTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem)

  Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.
- PowerLawTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩ System)

  Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope `[get, set]`

  The slope of the trendline (a).
- double Intercept `[get, set]`

  The intercept of the trendline (b).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

The maximum X value for which the trendline is plotted.

- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.

- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.

- string Tag `[get, set]`

  A tag to identify the trendline in the plot.

- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.120.1 Detailed Description

A plot element that draws a power law trendline with equation `y = b * x^a`.

Definition at line 877 of file Trendlines.cs.

7.120.2 Constructor & Destructor Documentation

7.120.2.1 PowerLawTrendLine() [1/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 935 of file Trendlines.cs.

7.120.2.2 PowerLawTrendLine() [2/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 951 of file Trendlines.cs.

7.120.2.3 PowerLawTrendLine() [3/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 985 of file Trendlines.cs.

7.120.3 Member Function Documentation

7.120.3.1 Plot()

```
void VectSharp.Plots.PowerLawTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 988 of file Trendlines.cs.

7.120.4 Property Documentation

7.120.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.PowerLawTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 922 of file Trendlines.cs.

7.120.4.2 Intercept

```
double VectSharp.Plots.PowerLawTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 887 of file Trendlines.cs.

7.120.4.3 MaxX

```
double VectSharp.Plots.PowerLawTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 902 of file Trendlines.cs.

7.120.4.4 MaxY

```
double VectSharp.Plots.PowerLawTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 907 of file Trendlines.cs.

7.120.4.5 MinX

```
double VectSharp.Plots.PowerLawTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 892 of file Trendlines.cs.

7.120.4.6 MinY

```
double VectSharp.Plots.PowerLawTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 897 of file Trendlines.cs.

7.120.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.PowerLawTrendLine.PresentationAttributes =
new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 912 of file Trendlines.cs.

7.120.4.8 Slope

```
double VectSharp.Plots.PowerLawTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 882 of file Trendlines.cs.

7.120.4.9 Tag

```
string VectSharp.Plots.PowerLawTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 917 of file Trendlines.cs.

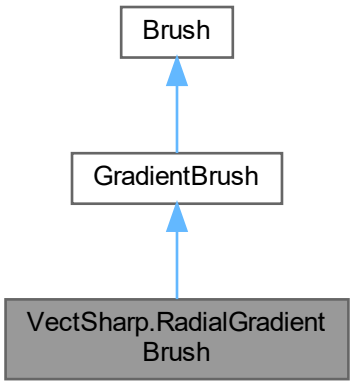
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.121 VectSharp.RadialGradientBrush Class Reference

Represents a brush painting with a radial gradient.

Inheritance diagram for VectSharp.RadialGradientBrush:



Public Member Functions

- RadialGradientBrush (Point focalPoint, Point centre, double radius, params GradientStop[ ] gradientStops)

  Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.
- RadialGradientBrush (Point focalPoint, Point centre, double radius, IEnumerable< GradientStop > gradient↩ Stops)

  Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Properties

- Point FocalPoint `[get]`

  The focal point of the gradient (i.e. the point within the circle where the gradient starts).
- Point Centre `[get]`

  Represents the centre of the gradient.
- double Radius `[get]`

  The radius of the gradient.

Additional Inherited Members

7.121.1 Detailed Description

Represents a brush painting with a radial gradient.

Definition at line 406 of file Brush.cs.

7.121.2 Constructor & Destructor Documentation

7.121.2.1 RadialGradientBrush() [1/2]

```
VectSharp.RadialGradientBrush.RadialGradientBrush (
            Point focalPoint,
            Point centre,
            double radius,
            params GradientStop[] gradientStops )
```

Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.

Parameters

focalPoint	The focal point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
centre	The centre of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
radius	The radius of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 430 of file Brush.cs.

7.121.2.2 RadialGradientBrush() [2/2]

```
VectSharp.RadialGradientBrush.RadialGradientBrush (
            Point focalPoint,
            Point centre,
            double radius,
            IEnumerable< GradientStop > gradientStops )
```

Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.

Parameters

focalPoint	The focal point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
centre	The centre of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
radius	The radius of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 469 of file Brush.cs.

7.121.3 Member Function Documentation

7.121.3.1 MultiplyOpacity()

```
override Brush VectSharp.RadialGradientBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 502 of file Brush.cs.

7.121.4 Property Documentation

7.121.4.1 Centre

```
Point VectSharp.RadialGradientBrush.Centre  [get]
```

Represents the centre of the gradient.

Definition at line 416 of file Brush.cs.

7.121.4.2 FocalPoint

```
Point VectSharp.RadialGradientBrush.FocalPoint  [get]
```

The focal point of the gradient (i.e. the point within the circle where the gradient starts).

Definition at line 411 of file Brush.cs.

7.121.4.3 Radius

```
double VectSharp.RadialGradientBrush.Radius  [get]
```

The radius of the gradient.

Definition at line 421 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.122 VectSharp.TrueTypeFile.CoverageTable.RangeRecord Struct Reference

Public Member Functions

- RangeRecord (Stream fs)

Public Attributes

- ushort StartGlyphID
- ushort EndGlyphID
- ushort StartCoverageIndex

7.122.1 Detailed Description

Definition at line 3802 of file TrueType.cs.

7.122.2 Constructor & Destructor Documentation

7.122.2.1 RangeRecord()

```
VectSharp.TrueTypeFile.CoverageTable.RangeRecord.RangeRecord (
            Stream fs )
```

Definition at line 3808 of file TrueType.cs.

7.122.3 Member Data Documentation

7.122.3.1 EndGlyphID

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.EndGlyphID
```

Definition at line 3805 of file TrueType.cs.

7.122.3.2 StartCoverageIndex

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.StartCoverageIndex
```

Definition at line 3806 of file TrueType.cs.

7.122.3.3 StartGlyphID

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.StartGlyphID
```

Definition at line 3804 of file TrueType.cs.

The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.123 VectSharp.Raster.Raster Class Reference

Contains methods to render a page to a PNG image.

Static Public Member Functions

- static void SaveAsPNG (this Page page, string fileName, double scale=1)

  Render the page to a PNG file.
- static void SaveAsPNG (this Page page, Stream stream, double scale=1)

  Render the page to a PNG stream.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.
- static DisposableIntPtr SaveAsRawBytes (this Page pag, out int width, out int height, out int totalSize, double scale=1)

  Render the page to raw pixel data, in 32bpp RGBA format.
- static void SaveAsAnimatedPNG (this Animation animation, Stream imageStream, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframeCompression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to a stream in animated PNG format.
- static void SaveAsAnimatedPNG (this Animation animation, string fileName, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframeCompression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to an animated PNG file.

7.123.1 Detailed Description

Contains methods to render a page to a PNG image.

Definition at line 30 of file Raster.cs.

7.123.2 Member Function Documentation

7.123.2.1 Rasterise()

```
static RasterImage VectSharp.Raster.Raster.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 91 of file Raster.cs.

7.123.2.2 SaveAsAnimatedPNG() [1/2]

```
static void VectSharp.Raster.Raster.SaveAsAnimatedPNG (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to a stream in animated PNG format.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated PNG will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 184 of file Raster.cs.

7.123.2.3 SaveAsAnimatedPNG() [2/2]

```
static void VectSharp.Raster.Raster.SaveAsAnimatedPNG (
            this Animation animation,
            string fileName,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to an animated PNG file.

Parameters

animation	The animation to export.
fileName	The output file to create.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 267 of file Raster.cs.

7.123.2.4 SaveAsPNG() [1/2]

```
static void VectSharp.Raster.Raster.SaveAsPNG (
            this Page page,
            Stream stream,
            double scale = 1 )  [static]
```

Render the page to a PNG stream.

Parameters

page	The Page to render.
stream	The stream to which the PNG data will be written.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image file.

Definition at line 64 of file Raster.cs.

7.123.2.5 SaveAsPNG() [2/2]

```
static void VectSharp.Raster.Raster.SaveAsPNG (
            this Page page,
            string fileName,
            double scale = 1 )  [static]
```

Render the page to a PNG file.

Parameters

page	The Page to render.
fileName	The full path to the file to save. If it exists, it will be overwritten.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image file.

Definition at line 39 of file Raster.cs.

7.123.2.6 SaveAsRawBytes()

```
static DisposableIntPtr VectSharp.Raster.Raster.SaveAsRawBytes (
            this Page pag,
            out int width,
            out int height,
            out int totalSize,
            double scale = 1 )  [static]
```

Render the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.
totalSize	The size in bytes of the raw pixel data.

Returns

A DisposableIntPtr containing a pointer to the raw pixel data, stored in unmanaged memory. Dispose this object to release the unmanaged memory.

Definition at line 143 of file Raster.cs.

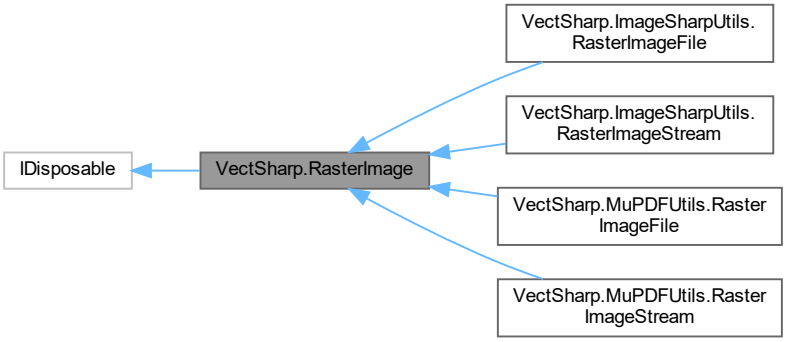
The documentation for this class was generated from the following file:

- VectSharp.Raster/Raster.cs

7.124 VectSharp.RasterImage Class Reference

Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.

Inheritance diagram for VectSharp.RasterImage:



Public Member Functions

- RasterImage (IntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)

  Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.
- RasterImage (ref DisposableIntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)

  Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.
- RasterImage (byte[ ] data, int width, int height, PixelFormats pixelFormat, bool interpolate)

  Creates a new RasterImage instance copying the specified pixel data.
- void ClearPNGCache ()

  Disposes the PNGStream. Also useful if is is necessary to regenerate it, e.g. because the underlying image pixel data has changed.
- void Dispose ()

Properties

- IntPtr ImageDataAddress `[get]`

  The memory address of the image pixel data.
- IDisposable DataHolder `[get]`

  An IDisposable that will be disposed when the image is disposed.
- string Id `[get]`

  A univocal identifier for this image.
- bool HasAlpha `[get]`

  Determines whether the image has an alpha channel.
- int Width `[get]`

  The width in pixels of the image.
- int Height `[get]`

  The height in pixels of the image.
- bool Interpolate `[get]`

  Determines whether the image should be interpolated when it is resized.
- MemoryStream PNGStream `[get]`

  Contains a representation of the image in PNG format. Generated at the first access and cached until the image is disposed.

7.124.1 Detailed Description

Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.

Definition at line 98 of file RasterImage.cs.

7.124.2 Constructor & Destructor Documentation

7.124.2.1 RasterImage() [1/3]

```
VectSharp.RasterImage.RasterImage (
            IntPtr pixelData,
            int width,
            int height,
            bool hasAlpha,
            bool interpolate )
```

Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.

Parameters

pixelData	The address of the image pixel data in RGB or RGBA format.
width	The width in pixels of the image.
height	The height in pixels of the image.
hasAlpha	true if the image is in RGBA format, false if it is in RGB format.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 170 of file RasterImage.cs.

7.124.2.2 RasterImage() [2/3]

```
VectSharp.RasterImage.RasterImage (
            ref DisposableIntPtr pixelData,
            int width,
            int height,
            bool hasAlpha,
            bool interpolate )
```

Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.

Parameters

pixelData	The address of the image pixel data in RGB or RGBA format wrapped in a DisposableIntPtr. The RasterImage will take ownership of this memory.

Parameters

width	The width in pixels of the image.
height	The height in pixels of the image.
hasAlpha	true if the image is in RGBA format, false if it is in RGB format.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 188 of file RasterImage.cs.

7.124.2.3 RasterImage() [3/3]

```
VectSharp.RasterImage.RasterImage (
            byte[] data,
            int width,
            int height,
            PixelFormats pixelFormat,
            bool interpolate )
```

Creates a new RasterImage instance copying the specified pixel data.

Parameters

data	The image pixel data that will be copied.
width	The width in pixels of the image.
height	The height in pixels of the image.
pixelFormat	The format of the pixel data.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 207 of file RasterImage.cs.

7.124.3 Member Function Documentation

7.124.3.1 ClearPNGCache()

```
void VectSharp.RasterImage.ClearPNGCache ( )
```

Disposes the PNGStream. Also useful if is is necessary to regenerate it, e.g. because the underlying image pixel data has changed.

Definition at line 261 of file RasterImage.cs.

7.124.3.2 Dispose()

```
void VectSharp.RasterImage.Dispose ( )
```

Definition at line 295 of file RasterImage.cs.

7.124.4 Property Documentation

7.124.4.1 DataHolder

```
IDisposable VectSharp.RasterImage.DataHolder  [get]
```

An IDisposable that will be disposed when the image is disposed.

Definition at line 108 of file RasterImage.cs.

7.124.4.2 HasAlpha

```
bool VectSharp.RasterImage.HasAlpha  [get]
```

Determines whether the image has an alpha channel.

Definition at line 118 of file RasterImage.cs.

7.124.4.3 Height

```
int VectSharp.RasterImage.Height  [get]
```

The height in pixels of the image.

Definition at line 128 of file RasterImage.cs.

7.124.4.4 Id

```
string VectSharp.RasterImage.Id  [get]
```

A univocal identifier for this image.

Definition at line 113 of file RasterImage.cs.

7.124.4.5 ImageDataAddress

```
IntPtr VectSharp.RasterImage.ImageDataAddress  [get]
```

The memory address of the image pixel data.

Definition at line 103 of file RasterImage.cs.

7.124.4.6 Interpolate

```
bool VectSharp.RasterImage.Interpolate  [get]
```

Determines whether the image should be interpolated when it is resized.

Definition at line 133 of file RasterImage.cs.

7.124.4.7 PNGStream

```
MemoryStream VectSharp.RasterImage.PNGStream  [get]
```

Contains a representation of the image in PNG format. Generated at the first access and cached until the image is disposed.

Definition at line 140 of file RasterImage.cs.

7.124.4.8 Width

```
int VectSharp.RasterImage.Width  [get]
```

The width in pixels of the image.

Definition at line 123 of file RasterImage.cs.

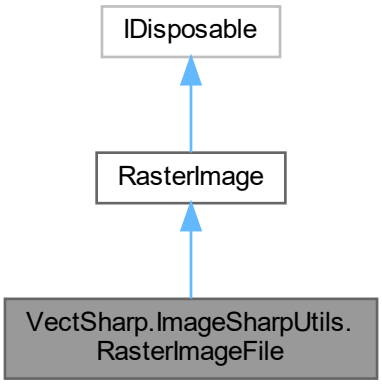
The documentation for this class was generated from the following file:

- VectSharp/RasterImage.cs

7.125 VectSharp.ImageSharpUtils.RasterImageFile Class Reference

A RasterImage created from a file.

Inheritance diagram for VectSharp.ImageSharpUtils.RasterImageFile:



Public Member Functions

- RasterImageFile (string fileName, bool alpha=true, bool interpolate=true)
  Creates a new RasterImage from the specified file.

Additional Inherited Members

7.125.1 Detailed Description

A RasterImage created from a file.

Definition at line 30 of file RasterImages.cs.

7.125.2 Constructor & Destructor Documentation

7.125.2.1 RasterImageFile()

```
VectSharp.ImageSharpUtils.RasterImageFile.RasterImageFile (
            string fileName,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified file.

Parameters

fileName	The path to the file containing the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 38 of file RasterImages.cs.

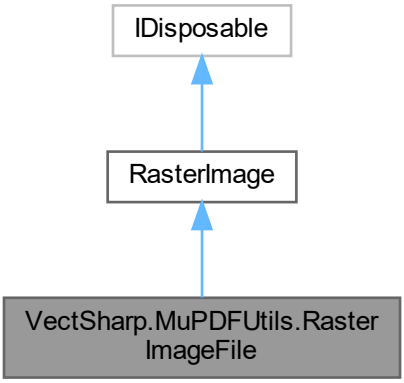
The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/RasterImages.cs

7.126 VectSharp.MuPDFUtils.RasterImageFile Class Reference

A RasterImage created from a file.

Inheritance diagram for VectSharp.MuPDFUtils.RasterImageFile:



Public Member Functions

- RasterImageFile (string fileName, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified file.

Additional Inherited Members

7.126.1 Detailed Description

A RasterImage created from a file.

Definition at line 28 of file RasterImages.cs.

7.126.2 Constructor & Destructor Documentation

7.126.2.1 RasterImageFile()

```
VectSharp.MuPDFUtils.RasterImageFile.RasterImageFile (
            string fileName,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified file.

Parameters

fileName	The path to the file containing the image.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 38 of file RasterImages.cs.

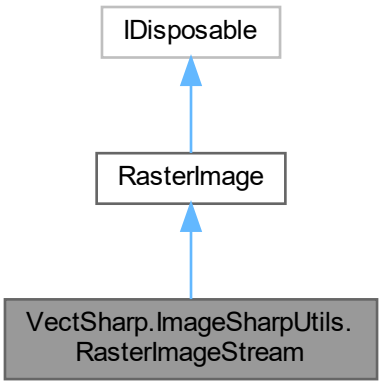
The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/RasterImages.cs

7.127 VectSharp.ImageSharpUtils.RasterImageStream Class Reference

A RasterImage created from a stream.

Inheritance diagram for VectSharp.ImageSharpUtils.RasterImageStream:



Public Member Functions

- RasterImageStream (Stream imageStream, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.
- RasterImageStream (IntPtr imageAddress, int imageLength, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.

Additional Inherited Members

7.127.1 Detailed Description

A RasterImage created from a stream.

Definition at line 106 of file RasterImages.cs.

7.127.2 Constructor & Destructor Documentation

7.127.2.1 RasterImageStream() [1/2]

```
VectSharp.ImageSharpUtils.RasterImageStream.RasterImageStream (
            Stream imageStream,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageStream	The stream containing the image data.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 114 of file RasterImages.cs.

7.127.2.2 RasterImageStream() [2/2]

```
VectSharp.ImageSharpUtils.RasterImageStream.RasterImageStream (
            IntPtr imageAddress,
            int imageLength,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageAddress	A pointer to the address where the image data is contained.
imageLength	The length in bytes of the image data.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 185 of file RasterImages.cs.

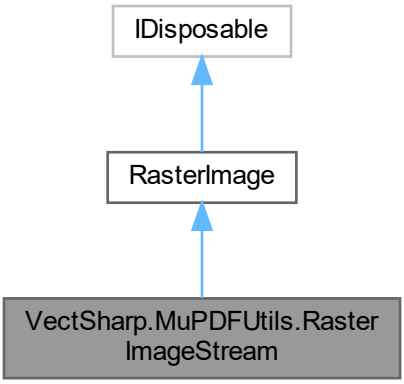
The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/RasterImages.cs

7.128 VectSharp.MuPDFUtils.RasterImageStream Class Reference

A RasterImage created from a stream.

Inheritance diagram for VectSharp.MuPDFUtils.RasterImageStream:



Public Member Functions

- RasterImageStream (Stream imageStream, InputFileTypes fileType, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.
- RasterImageStream (IntPtr imageAddress, long imageLength, InputFileTypes fileType, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.

Additional Inherited Members

7.128.1 Detailed Description

A RasterImage created from a stream.

Definition at line 69 of file RasterImages.cs.

7.128.2 Constructor & Destructor Documentation

7.128.2.1 RasterImageStream() [1/2]

```
VectSharp.MuPDFUtils.RasterImageStream.RasterImageStream (
            Stream imageStream,
            InputFileTypes fileType,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageStream	The stream containing the image data.
fileType	The type of the image contained in the stream.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 80 of file RasterImages.cs.

7.128.2.2 RasterImageStream() [2/2]

```
VectSharp.MuPDFUtils.RasterImageStream.RasterImageStream (
            IntPtr imageAddress,
            long imageLength,
            InputFileTypes fileType,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageAddress	A pointer to the address where the image data is contained.
imageLength	The length in bytes of the image data.
fileType	The type of the image contained in the stream.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 148 of file RasterImages.cs.

The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/RasterImages.cs

7.129 VectSharp.Rectangle Struct Reference

Represents a rectangle.

Public Member Functions

- bool IsNaN ()

  Checks whether the rectangle is equivalent to Rectangle.NaN.
- Rectangle (Point location, Size size)

  Create a new Rectangle given its top-left corner and its size.
- Rectangle (double x, double y, double width, double height)

  Create a new Rectangle given its top-left corner and its size.
- Rectangle (Point topLeft, Point bottomRight)

  Create a new Rectangle given its top-left corner and its bottom-right corner.

Static Public Member Functions

- static Rectangle Union (Rectangle rectangle1, Rectangle rectangle2)

  Computes the rectangular bounds of the union of two Rectangles.
- static Rectangle Intersection (Rectangle rectangle1, Rectangle rectangle2)

  Computes the intersection of two Rectangles.
- static Rectangle Union (IEnumerable< Rectangle > rectangles)

  Computes the rectangular bounds of the union of multiple Rectangles.
- static Rectangle Union (params Rectangle[ ] rectangles)

  Computes the rectangular bounds of the union of multiple Rectangles.

Public Attributes

- Point Location

  The top-left corner of the rectangle.
- Size Size

  The size of the rectangle.

Static Public Attributes

- static readonly Rectangle NaN = new Rectangle(double.NaN, double.NaN, double.NaN, double.NaN)

  A rectangle whose dimensions are all double.NaN.

Properties

- Point Centre `[get]`

  The centre of the rectangle.

7.129.1 Detailed Description

Represents a rectangle.

Definition at line 295 of file Point.cs.

7.129.2 Constructor & Destructor Documentation

7.129.2.1 Rectangle() [1/3]

```
VectSharp.Rectangle.Rectangle (
            Point location,
            Size size )
```

Create a new Rectangle given its top-left corner and its size.

Parameters

location	The top-left corner of the rectangle.
size	The size of the rectangle.

Definition at line 331 of file Point.cs.

7.129.2.2 Rectangle() [2/3]

```
VectSharp.Rectangle.Rectangle (
            double x,
            double y,
            double width,
            double height )
```

Create a new Rectangle given its top-left corner and its size.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle.
y	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.

Definition at line 344 of file Point.cs.

7.129.2.3 Rectangle() [3/3]

```
VectSharp.Rectangle.Rectangle (
            Point topLeft,
            Point bottomRight )
```

Create a new Rectangle given its top-left corner and its bottom-right corner.

Parameters

topLeft	The top-left corner of the rectangle.
bottomRight	The bottom-right corner of the rectangle.

Definition at line 355 of file Point.cs.

7.129.3 Member Function Documentation

7.129.3.1 Intersection()

```
static Rectangle VectSharp.Rectangle.Intersection (
            Rectangle rectangle1,
            Rectangle rectangle2 )  [static]
```

Computes the intersection of two Rectangles.

Parameters

rectangle1	The first Rectangle.
rectangle2	The second Rectangle.

Returns

The rectangle corresponding to the intersection of *rectangle1* and *rectangle2* , or Rectangle.NaN if the intersection is empty.

Definition at line 400 of file Point.cs.

7.129.3.2 IsNaN()

```
bool VectSharp.Rectangle.IsNaN ( )
```

Checks whether the rectangle is equivalent to Rectangle.NaN.

Returns

`true` if all the dimensions of the rectangle are double.NaN, `false` otherwise.

Definition at line 306 of file Point.cs.

7.129.3.3 Union() [1/3]

```
static Rectangle VectSharp.Rectangle.Union (
            IEnumerable< Rectangle > rectangles )  [static]
```

Computes the rectangular bounds of the union of multiple Rectangles.

Parameters

rectangles	The Rectangles whose union will be computed.

Returns

The smallest Rectangle containing all the *rectangles* .

Definition at line 424 of file Point.cs.

7.129.3.4 Union() [2/3]

```
static Rectangle VectSharp.Rectangle.Union (
            params Rectangle[] rectangles )  [static]
```

Computes the rectangular bounds of the union of multiple Rectangles.

Parameters

rectangles	The Rectangles whose union will be computed.

Returns

The smallest Rectangle containing all the *rectangles* .

Definition at line 457 of file Point.cs.

7.129.3.5 Union() [3/3]

```
static Rectangle VectSharp.Rectangle.Union (
            Rectangle rectangle1,
            Rectangle rectangle2 )  [static]
```

Computes the rectangular bounds of the union of two Rectangles.

Parameters

rectangle1	The first Rectangle.
rectangle2	The second Rectangle.

Returns

The smallest Rectangle containing both *rectangle1* and *rectangle2* .

Definition at line 367 of file Point.cs.

7.129.4 Member Data Documentation

7.129.4.1 Location

```
Point VectSharp.Rectangle.Location
```

The top-left corner of the rectangle.

Definition at line 314 of file Point.cs.

7.129.4.2 NaN

```
readonly Rectangle VectSharp.Rectangle.NaN = new Rectangle(double.NaN, double.NaN, double.NaN,
double.NaN)  [static]
```

A rectangle whose dimensions are all double.NaN.

Definition at line 300 of file Point.cs.

7.129.4.3 Size

```
Size VectSharp.Rectangle.Size
```

The size of the rectangle.

Definition at line 319 of file Point.cs.

7.129.5 Property Documentation

7.129.5.1 Centre

```
Point VectSharp.Rectangle.Centre  [get]
```

The centre of the rectangle.

Definition at line 324 of file Point.cs.

The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.130 VectSharp.Canvas.RenderAction Class Reference

Represents a light-weight rendering action.

Public Types

- enum ActionTypes

  Types of rendering actions.

Public Member Functions

- void BringToFront ()

  Brings the render action to the front of the rendering queue. This method can only be invoked after the output has been fully initialised.
- void SendToBack ()

  Brings the render action to the back of the rendering queue. This method can only be invoked after the output has been fully initialised.

Static Public Member Functions

- static RenderAction PathAction (Geometry geometry, Pen stroke, IBrush fill, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing a path.
- static RenderAction TextAction (Avalonia.Media.FormattedText text, Avalonia.Media.TextFormatting.Text↩ Layout layout, IBrush fill, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing text.
- static RenderAction ImageAction (string imageId, Avalonia.Rect sourceRect, Avalonia.Rect destinationRect, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing an image.

Properties

- ActionTypes ActionType `[get]`

  Type of the rendering action.
- Geometry Geometry `[get, set]`

  Geometry that needs to be rendered (null if the action type is ActionTypes.Text). If you change this, you need to invalidate the Parent's visual.
- Avalonia.Media.FormattedText Text `[get, set]`

  Text that needs to be rendered (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Layout property.
- Avalonia.Media.TextFormatting.TextLayout Layout `[get, set]`

  Text layout, used for hit testing (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Text property.
- Pen Stroke `[get, set]`

  Rendering stroke (null if the action type is ActionTypes.Text or if the rendered action only has a Fill). If you change this, you need to invalidate the Parent's visual.
- IBrush Fill `[get, set]`

  Rendering fill (null if the rendered action only has a Stroke). If you change this, you need to invalidate the Parent's visual.
- string ImageId `[get, set]`

  Univocal identifier of the image that needs to be drawn.
- Avalonia.? Rect ImageSource `[get, set]`

  The source rectangle of the image.
- Avalonia.? Rect ImageDestination `[get, set]`

  The destination rectangle of the image.

- Geometry ClippingPath [get, set]

  The current clipping path.
- Avalonia.Matrix InverseTransform = Avalonia.Matrix.Identity [get]

  Inverse transformation matrix.
- Avalonia.Matrix Transform [get, set]

  Rendering transformation matrix. If you change this, you need to invalidate the Parent's visual.
- string Tag [get, set]

  A tag to access the RenderAction.
- Control Parent [get]

  The container of this RenderAction.

Events

- EventHandler< Avalonia.Input.PointerEventArgs > PointerEntered

  Raised when the pointer enters the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerEventArgs > PointerExited

  Raised when the pointer leaves the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerPressedEventArgs > PointerPressed

  Raised when the pointer is pressed while over the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerReleasedEventArgs > PointerReleased

  Raised when the pointer is released after a PointerPressed event.

7.130.1 Detailed Description

Represents a light-weight rendering action.

Definition at line 1355 of file AvaloniaContext.cs.

7.130.2 Member Enumeration Documentation

7.130.2.1 ActionTypes

```
enum VectSharp.Canvas.RenderAction.ActionTypes
```

Types of rendering actions.

Definition at line 1360 of file AvaloniaContext.cs.

7.130.3 Member Function Documentation

7.130.3.1 BringToFront()

```
void VectSharp.Canvas.RenderAction.BringToFront ( )
```

Brings the render action to the front of the rendering queue. This method can only be invoked after the output has been fully initialised.

Definition at line 1608 of file AvaloniaContext.cs.

7.130.3.2 ImageAction()

```
static RenderAction VectSharp.Canvas.RenderAction.ImageAction (
            string imageId,
            Avalonia.Rect sourceRect,
            Avalonia.Rect destinationRect,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing an image.

Parameters

imageId	The univocal identifier of the image to draw.
sourceRect	The source rectangle of the image.
destinationRect	The destination rectangle of the image.
transform	The transform that will be applied to the image.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing an image.

Definition at line 1591 of file AvaloniaContext.cs.

7.130.3.3 PathAction()

```
static RenderAction VectSharp.Canvas.RenderAction.PathAction (
            Geometry geometry,
            Pen stroke,
            IBrush fill,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing a path.

Parameters

geometry	The geometry to be rendered.
stroke	The stroke of the path (can be null).
fill	The fill of the path (can be null).
transform	The transform that will be applied to the path.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing a path.

Definition at line 1542 of file AvaloniaContext.cs.

7.130.3.4 SendToBack()

```
void VectSharp.Canvas.RenderAction.SendToBack ( )
```

Brings the render action to the back of the rendering queue. This method can only be invoked after the output has been fully initialised.

Definition at line 1616 of file AvaloniaContext.cs.

7.130.3.5 TextAction()

```
static RenderAction VectSharp.Canvas.RenderAction.TextAction (
            Avalonia.Media.FormattedText text,
            Avalonia.Media.TextFormatting.TextLayout layout,
            IBrush fill,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing text.

Parameters

text	The text to be rendered.
layout	The text layout used for hit testing.
fill	The fill of the text (can be null).
transform	The transform that will be applied to the text.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing text.

Definition at line 1566 of file AvaloniaContext.cs.

7.130.4 Property Documentation

7.130.4.1 ActionType

```
ActionTypes VectSharp.Canvas.RenderAction.ActionType  [get]
```

Type of the rendering action.

Definition at line 1381 of file AvaloniaContext.cs.

7.130.4.2 ClippingPath

```
Geometry VectSharp.Canvas.RenderAction.ClippingPath  [get], [set]
```

The current clipping path.

Definition at line 1446 of file AvaloniaContext.cs.

7.130.4.3 Fill

```
IBrush VectSharp.Canvas.RenderAction.Fill  [get], [set]
```

Rendering fill (null if the rendered action only has a Stroke). If you change this, you need to invalidate the Parent's visual.

Definition at line 1411 of file AvaloniaContext.cs.

7.130.4.4 Geometry

```
Geometry VectSharp.Canvas.RenderAction.Geometry  [get], [set]
```

Geometry that needs to be rendered (null if the action type is ActionTypes.Text). If you change this, you need to invalidate the Parent's visual.

Definition at line 1386 of file AvaloniaContext.cs.

7.130.4.5 ImageDestination

```
Avalonia.?  Rect VectSharp.Canvas.RenderAction.ImageDestination  [get], [set]
```

The destination rectangle of the image.

Definition at line 1441 of file AvaloniaContext.cs.

7.130.4.6 ImageId

```
string VectSharp.Canvas.RenderAction.ImageId  [get], [set]
```

Univocal identifier of the image that needs to be drawn.

Definition at line 1431 of file AvaloniaContext.cs.

7.130.4.7 ImageSource

```
Avalonia.?  Rect VectSharp.Canvas.RenderAction.ImageSource  [get], [set]
```

The source rectangle of the image.

Definition at line 1436 of file AvaloniaContext.cs.

7.130.4.8 InverseTransform

```
Avalonia.Matrix VectSharp.Canvas.RenderAction.InverseTransform = Avalonia.Matrix.Identity
[get]
```

Inverse transformation matrix.

Definition at line 1453 of file AvaloniaContext.cs.

7.130.4.9 Layout

```
Avalonia.Media.TextFormatting.TextLayout VectSharp.Canvas.RenderAction.Layout  [get], [set]
```

Text layout, used for hit testing (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Text property.

Definition at line 1398 of file AvaloniaContext.cs.

7.130.4.10 Parent

```
Control VectSharp.Canvas.RenderAction.Parent  [get]
```

The container of this RenderAction.

Definition at line 1478 of file AvaloniaContext.cs.

7.130.4.11 Stroke

```
Pen VectSharp.Canvas.RenderAction.Stroke  [get], [set]
```

Rendering stroke (null if the action type is ActionTypes.Text or if the rendered action only has a Fill). If you change this, you need to invalidate the Parent's visual.

Definition at line 1403 of file AvaloniaContext.cs.

7.130.4.12 Tag

```
string VectSharp.Canvas.RenderAction.Tag  [get], [set]
```

A tag to access the RenderAction.

Definition at line 1471 of file AvaloniaContext.cs.

7.130.4.13 Text

```
Avalonia.Media.FormattedText VectSharp.Canvas.RenderAction.Text  [get], [set]
```

Text that needs to be rendered (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Layout property.

Definition at line 1392 of file AvaloniaContext.cs.

7.130.4.14 Transform

```
Avalonia.Matrix VectSharp.Canvas.RenderAction.Transform  [get], [set]
```

Rendering transformation matrix. If you change this, you need to invalidate the Parent's visual.

Definition at line 1458 of file AvaloniaContext.cs.

7.130.5 Event Documentation

7.130.5.1 PointerEntered

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.RenderAction.PointerEntered
```

Raised when the pointer enters the area covered by the RenderAction.

Definition at line 1489 of file AvaloniaContext.cs.

7.130.5.2 PointerExited

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.RenderAction.PointerExited
```

Raised when the pointer leaves the area covered by the RenderAction.

Definition at line 1494 of file AvaloniaContext.cs.

7.130.5.3 PointerPressed

```
EventHandler<Avalonia.Input.PointerPressedEventArgs> VectSharp.Canvas.RenderAction.Pointer↩
Pressed
```

Raised when the pointer is pressed while over the area covered by the RenderAction.

Definition at line 1499 of file AvaloniaContext.cs.

7.130.5.4 PointerReleased

```
EventHandler<Avalonia.Input.PointerReleasedEventArgs> VectSharp.Canvas.RenderAction.Pointer↩
Released
```

Raised when the pointer is released after a PointerPressed event.

Definition at line 1504 of file AvaloniaContext.cs.

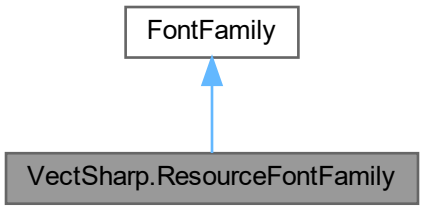
The documentation for this class was generated from the following file:

- VectSharp.Canvas/AvaloniaContext.cs

7.131 VectSharp.ResourceFontFamily Class Reference

Represents a FontFamily created from a resource stream.

Inheritance diagram for VectSharp.ResourceFontFamily:



Public Member Functions

- ResourceFontFamily (System.IO.Stream resourceStream, string resourceName)

  Create a new ResourceFontFamily from the specified resourceStream containing a TTF file, passing the specified resourceName to the

Public Attributes

- string ResourceName

  The name of the embedded resource, which will be parsed using

Additional Inherited Members

7.131.1 Detailed Description

Represents a FontFamily created from a resource stream.

Definition at line 694 of file Font.cs.

7.131.2 Constructor & Destructor Documentation

7.131.2.1 ResourceFontFamily()

```
VectSharp.ResourceFontFamily.ResourceFontFamily (
            System.IO.Stream resourceStream,
            string resourceName )
```

Create a new ResourceFontFamily from the specified *resourceStream* containing a TTF file, passing the specified *resourceName* to the

`Avalonia.Media.FontFamily.Parse(string, Uri)"</code> method. </summary>
<param name="resourceStream">A resource stream containing a TTF file.</param>
<param name="resourceName">The name of the embedded resource, which will be parsed using Avalonia.Media.FontFamily.Parse(string, Uri).`

Definition at line 706 of file Font.cs.

7.131.3 Member Data Documentation

7.131.3.1 ResourceName

```
string VectSharp.ResourceFontFamily.ResourceName
```

The name of the embedded resource, which will be parsed using

`Avalonia.Media.FontFamily.Parse(string, Uri).`

Definition at line 699 of file Font.cs.

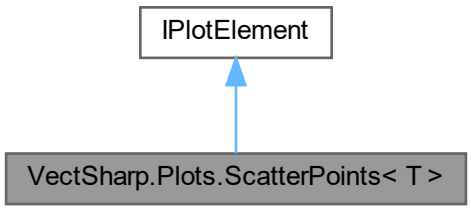
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.132 VectSharp.Plots.ScatterPoints< T > Class Template Reference

A plot element that draws a symbol at the location of multiple data points.

Inheritance diagram for VectSharp.Plots.ScatterPoints< T >:



Public Member Functions

- ScatterPoints (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Creates a new ScatterPoints< T > instance, using the specified data and coordinateSystem .
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points at which the symbols will be drawn.
- double Size = 2 `[get, set]`

  The size of the symbols to draw, in plot coordinates.
- IDataPointElement DataPointElement = new PathDataPointElement() `[get, set]`

  The symbol that will be drawn (by default, a circle).
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance (stroke and fill colour, etc.) of the symbols.
- string Tag `[get, set]`

  A tag to identify the symbols in the plot.

7.132.1 Detailed Description

A plot element that draws a symbol at the location of multiple data points.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 150 of file DataPoints.cs.

7.132.2 Constructor & Destructor Documentation

7.132.2.1 ScatterPoints()

```
VectSharp.Plots.ScatterPoints< T >.ScatterPoints (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Creates a new ScatterPoints<T> instance, using the specified *data* and *coordinateSystem* .

Parameters

data	The data points at which the symbols will be drawn.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 188 of file DataPoints.cs.

7.132.3 Member Function Documentation

7.132.3.1 Plot()

```
void VectSharp.Plots.ScatterPoints< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 195 of file DataPoints.cs.

7.132.4 Property Documentation

7.132.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.ScatterPoints< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 170 of file DataPoints.cs.

7.132.4.2 Data

```
IEnumerable<T> VectSharp.Plots.ScatterPoints< T >.Data  [get], [set]
```

The data points at which the symbols will be drawn.

Definition at line 155 of file DataPoints.cs.

7.132.4.3 DataPointElement

```
IDataPointElement VectSharp.Plots.ScatterPoints< T >.DataPointElement = new PathDataPointElement()
[get], [set]
```

The symbol that will be drawn (by default, a circle).

Definition at line 165 of file DataPoints.cs.

7.132.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ScatterPoints< T >.PresentationAttributes =
new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance (stroke and fill colour, etc.) of the symbols.

Definition at line 176 of file DataPoints.cs.

7.132.4.5 Size

```
double VectSharp.Plots.ScatterPoints< T >.Size = 2  [get], [set]
```

The size of the symbols to draw, in plot coordinates.

Definition at line 160 of file DataPoints.cs.

7.132.4.6 Tag

```
string VectSharp.Plots.ScatterPoints< T >.Tag  [get], [set]
```

A tag to identify the symbols in the plot.

Definition at line 181 of file DataPoints.cs.

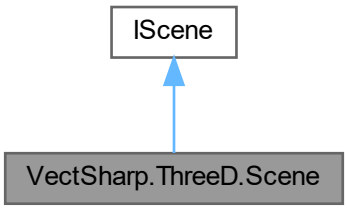
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.133 VectSharp.ThreeD.Scene Class Reference

Represents a 3D scene.

Inheritance diagram for VectSharp.ThreeD.Scene:



Public Member Functions

- Scene ()

  Creates a new Scene.
- void AddElement (Element3D element)

  Adds the specified element to the scene.

Parameters

element	*The Element3D to add.*

- void AddRange (IEnumerable< Element3D > elements)

  Adds the specified elements to the scene.

Parameters

elements	*A collection of Element3Ds to add.*

- void Replace (Func< Element3D, Element3D > replacementFunction)

  Replaces each element in the scene with the element returned by the replacementFunction .

Parameters

replacementFunction	*A function replacing each Element3D in the scene with another Element3D.*

- void Replace (Func< Element3D, IEnumerable< Element3D > > replacementFunction)

  Replaces each element in the scene with the element(s) returned by the replacementFunction .

Parameters

replacementFunction	*A function replacing each Element3D in the scene with 0 or more Element3Ds.*

Properties

- object SceneLock [get]

  An object used to synchronise multithreaded rendering of the same scene.
- IEnumerable< Element3D > SceneElements [get]

  The Element3Ds constituting the scene.

7.133.1 Detailed Description

Represents a 3D scene.

Definition at line 66 of file Scene.cs.

7.133.2 Constructor & Destructor Documentation

7.133.2.1 Scene()

```
VectSharp.ThreeD.Scene.Scene ( )
```

Creates a new Scene.

Definition at line 79 of file Scene.cs.

7.133.3 Member Function Documentation

7.133.3.1 AddElement()

```
void VectSharp.ThreeD.Scene.AddElement (
            Element3D element )
```

Adds the specified *element* to the scene.

Parameters

element	The Element3D to add.

Implements VectSharp.ThreeD.IScene.

Definition at line 86 of file Scene.cs.

7.133.3.2 AddRange()

```
void VectSharp.ThreeD.Scene.AddRange (
            IEnumerable< Element3D > elements )
```

Adds the specified *elements* to the scene.

Parameters

elements	A collection of Element3Ds to add.

Implements VectSharp.ThreeD.IScene.

Definition at line 92 of file Scene.cs.

7.133.3.3 Replace() [1/2]

```
void VectSharp.ThreeD.Scene.Replace (
            Func< Element3D, Element3D > replacementFunction )
```

Replaces each element in the scene with the element returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with another Element3D.

Implements VectSharp.ThreeD.IScene.

Definition at line 98 of file Scene.cs.

7.133.3.4 Replace() [2/2]

```
void VectSharp.ThreeD.Scene.Replace (
            Func< Element3D, IEnumerable< Element3D > > replacementFunction )
```

Replaces each element in the scene with the element(s) returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with 0 or more Element3Ds.

Implements VectSharp.ThreeD.IScene.

Definition at line 107 of file Scene.cs.

7.133.4 Property Documentation

7.133.4.1 SceneElements

```
IEnumerable<Element3D> VectSharp.ThreeD.Scene.SceneElements  [get]
```

The Element3Ds constituting the scene.

Implements VectSharp.ThreeD.IScene.

Definition at line 74 of file Scene.cs.

7.133.4.2 SceneLock

```
object VectSharp.ThreeD.Scene.SceneLock  [get]
```

An object used to synchronise multithreaded rendering of the same scene.

Implements VectSharp.ThreeD.IScene.

Definition at line 69 of file Scene.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Scene.cs

7.134 VectSharp.Segment Class Reference

Represents a segment as part of a GraphicsPath.

Public Member Functions

- abstract Segment Clone ()

  Creates a copy of the Segment.
- abstract double Measure (Point previousPoint)

  Computes the length of the Segment.
- abstract Point GetPointAt (Point previousPoint, double position)

  Gets the point on the Segment at the specified (relative) position).
- abstract Point GetTangentAt (Point previousPoint, double position)

  Gets the tangent to the Segment at the specified (relative) position).
- abstract IEnumerable< Segment > Linearise (Point? previousPoint, double resolution)

  Transform the segment into a series of linear segments. Segments that are already linear are not changed.
- abstract IEnumerable< Point > GetLinearisationTangents (Point? previousPoint, double resolution)

  Gets the tanget at the points at which the segment would be linearised.
- abstract IEnumerable< Segment > Transform (Func< Point, Point > transformationFunction)

  Applies an arbitrary transformation to all of the points of the Segment.
- abstract IEnumerable< Segment > Flatten (Point? previousPoint, double flatness)

  Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.
- abstract IEnumerable<(Point point, Point tangent)> FlattenForOffsetAndGetTangents (Point? previousPoint, double offset, double flatness)

  Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path, assuming that the Segment will be drawn with the specified offset .

Properties

- abstract SegmentType Type [get]

  The type of the Segment.
- Point[] Points [get]

  The points used to define the Segment.
- virtual Point Point [get]

  The end point of the Segment.

7.134.1 Detailed Description

Represents a segment as part of a GraphicsPath.

Definition at line 28 of file Segment.cs.

7.134.2 Member Function Documentation

7.134.2.1 Clone()

```
abstract Segment VectSharp.Segment.Clone ( )  [pure virtual]
```

Creates a copy of the Segment.

Returns

A copy of the Segment.

7.134.2.2 Flatten()

```
abstract IEnumerable< Segment > VectSharp.Segment.Flatten (
            Point?  previousPoint,
            double flatness )  [pure virtual]
```

Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
flatness	The maximum deviation from the original path.

Returns

A collection of Segments composed only of linear segments that approximates the current Segment.

7.134.2.3 FlattenForOffsetAndGetTangents()

```
abstract IEnumerable<(Point point, Point tangent)> VectSharp.Segment.FlattenForOffsetAndGet↩
Tangents (
            Point?  previousPoint,
            double offset,
            double flatness )  [pure virtual]
```

Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path, assuming that the Segment will be drawn with the specified *offset* .

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
offset	The offset that will be used to draw the Segment (e.g., the line width of the stroke).
flatness	The maximum deviation from the original path.

Returns

A collection of tuples where the first element is a Point representing the end-point of a linear segment, and the second element is a Point containing the value of the tangent to the original Segment at that point.

7.134.2.4 GetLinearisationTangents()

```
abstract IEnumerable< Point > VectSharp.Segment.GetLinearisationTangents (
            Point?  previousPoint,
            double resolution )  [pure virtual]
```

Gets the tanget at the points at which the segment would be linearised.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
resolution	The absolute length between successive samples in curve segments.

Returns

A collection of tangents at the points in which the segment would be linearised.

7.134.2.5 GetPointAt()

```
abstract Point VectSharp.Segment.GetPointAt (
            Point previousPoint,
            double position )  [pure virtual]
```

Gets the point on the Segment at the specified (relative) *position*).

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
position	The relative position on the Segment (0 is the start of the Segment, 1 is the end of the Segment).

Returns

The point at the specified position.

7.134.2.6 GetTangentAt()

```
abstract Point VectSharp.Segment.GetTangentAt (
            Point previousPoint,
            double position )  [pure virtual]
```

Gets the tangent to the Segment at the specified (relative) *position*).

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
position	The relative position on the Segment (0 is the start of the Segment, 1 is the end of the Segment).

Returns

The tangent to the point at the specified position.

7.134.2.7 Linearise()

```
abstract IEnumerable< Segment > VectSharp.Segment.Linearise (
            Point?  previousPoint,
            double resolution )  [pure virtual]
```

Transform the segment into a series of linear segments. Segments that are already linear are not changed.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
resolution	The absolute length between successive samples in curve segments.

Returns

A collection of linear segments that approximate the current segment.

7.134.2.8 Measure()

```
abstract double VectSharp.Segment.Measure (
            Point previousPoint )  [pure virtual]
```

Computes the length of the Segment.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).

Returns

The length of the segment.

7.134.2.9 Transform()

```
abstract IEnumerable< Segment > VectSharp.Segment.Transform (
            Func< Point, Point > transformationFunction )  [pure virtual]
```

Applies an arbitrary transformation to all of the points of the Segment.

Parameters

transformationFunction	An arbitrary transformation function.

Returns

A collection of Segments that have been transformed according to the *transformationFunction* .

7.134.3 Property Documentation

7.134.3.1 Point

```
virtual Point VectSharp.Segment.Point  [get]
```

The end point of the Segment.

Definition at line 44 of file Segment.cs.

7.134.3.2 Points

```
Point [] VectSharp.Segment.Points  [get]
```

The points used to define the Segment.

Definition at line 39 of file Segment.cs.

7.134.3.3 Type

```
abstract SegmentType VectSharp.Segment.Type  [get]
```

The type of the Segment.

Definition at line 34 of file Segment.cs.

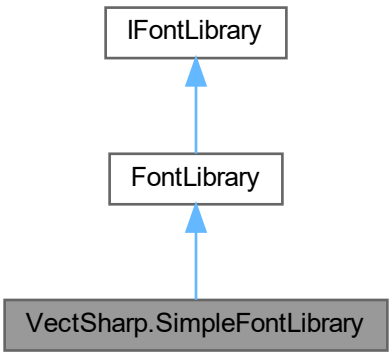
The documentation for this class was generated from the following file:

- VectSharp/Segment.cs

7.135 VectSharp.SimpleFontLibrary Class Reference

A font library that can be used to cache and resolve font family names.

Inheritance diagram for VectSharp.SimpleFontLibrary:



Public Member Functions

- SimpleFontLibrary (IFontLibrary standardFontLibrary)

  Create a new SimpleFontLibrary instance.
- SimpleFontLibrary ()

  Create a new SimpleFontLibrary instance, using the default font library to resolve the standard font families.
- SimpleFontLibrary (FontFamily timesRoman, FontFamily timesBold, FontFamily timesItalic, FontFamily timesBoldItalic, FontFamily helvetica, FontFamily helveticaBold, FontFamily helveticaOblique, FontFamily helveticaBoldOblique, FontFamily courier, FontFamily courierBold, FontFamily courierOblique, FontFamily courierBoldOblique, FontFamily symbol, FontFamily zapfdingbats)

  Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.
- SimpleFontLibrary (string timesRoman, string timesBold, string timesItalic, string timesBoldItalic, string helvetica, string helveticaBold, string helveticaOblique, string helveticaBoldOblique, string courier, string courierBold, string courierOblique, string courierBoldOblique, string symbol, string zapfdingbats)

  Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.
- void Add (string fontFamilyName, FontFamily fontFamily)

  Add the specified font family to the library with the specified name.
- void Add (FontFamily fontFamily)

  Add the specified font family to the library.
- void Add (string fileName)

  Add the font family contained in the specified True Type Font file to the library.
- void Add (string fontFamily, string fileName)

  Add the font family contained in the specified True Type Font file to the library, with the specified name. The font family is not loaded until it is requested for the first time.
- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

7.135.1 Detailed Description

A font library that can be used to cache and resolve font family names.

Definition at line 185 of file FontLibrary.cs.

7.135.2 Constructor & Destructor Documentation

7.135.2.1 SimpleFontLibrary() [1/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            IFontLibrary standardFontLibrary )
```

Create a new SimpleFontLibrary instance.

Parameters

standardFontLibrary	An existing font library that will be used to resolve the standard font families.

Definition at line 197 of file FontLibrary.cs.

7.135.2.2 SimpleFontLibrary() [2/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary ( )
```

Create a new SimpleFontLibrary instance, using the default font library to resolve the standard font families.

Definition at line 217 of file FontLibrary.cs.

7.135.2.3 SimpleFontLibrary() [3/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            FontFamily timesRoman,
            FontFamily timesBold,
            FontFamily timesItalic,
            FontFamily timesBoldItalic,
            FontFamily helvetica,
            FontFamily helveticaBold,
            FontFamily helveticaOblique,
            FontFamily helveticaBoldOblique,
            FontFamily courier,
            FontFamily courierBold,
            FontFamily courierOblique,
            FontFamily courierBoldOblique,
            FontFamily symbol,
            FontFamily zapfdingbats )
```

Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.

Parameters

timesRoman	The font family to use for the Times-Roman standard font.
timesBold	The font family to use for the Times-Bold standard font.
timesItalic	The font family to use for the Times-Italic standard font.
timesBoldItalic	The font family to use for the Times-BoldItalic standard font.
helvetica	The font family to use for the Helvetica standard font.

Parameters

helveticaBold	The font family to use for the Helvetica-Bold standard font.
helveticaOblique	The font family to use for the Helvetica-Oblique standard font.
helveticaBoldOblique	The font family to use for the Helvetica-BoldOblique standard font.
courier	The font family to use for the Courier standard font.
courierBold	The font family to use for the Courier-Bold standard font.
courierOblique	The font family to use for the Courier-Oblique standard font.
courierBoldOblique	The font family to use for the Courier-BoldOblique standard font.
symbol	The font family to use for the Symbol standard font.
zapfdingbats	The font family to use for the Zapfdingbats standard font.

Definition at line 239 of file FontLibrary.cs.

7.135.2.4 SimpleFontLibrary() [4/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            string timesRoman,
            string timesBold,
            string timesItalic,
            string timesBoldItalic,
            string helvetica,
            string helveticaBold,
            string helveticaOblique,
            string helveticaBoldOblique,
            string courier,
            string courierBold,
            string courierOblique,
            string courierBoldOblique,
            string symbol,
            string zapfdingbats )
```

Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.

Parameters

timesRoman	The font family to use for the Times-Roman standard font.
timesBold	The font family to use for the Times-Bold standard font.
timesItalic	The font family to use for the Times-Italic standard font.
timesBoldItalic	The font family to use for the Times-BoldItalic standard font.
helvetica	The font family to use for the Helvetica standard font.
helveticaBold	The font family to use for the Helvetica-Bold standard font.
helveticaOblique	The font family to use for the Helvetica-Oblique standard font.
helveticaBoldOblique	The font family to use for the Helvetica-BoldOblique standard font.
courier	The font family to use for the Courier standard font.
courierBold	The font family to use for the Courier-Bold standard font.
courierOblique	The font family to use for the Courier-Oblique standard font.
courierBoldOblique	The font family to use for the Courier-BoldOblique standard font.
symbol	The font family to use for the Symbol standard font.
zapfdingbats	The font family to use for the Zapfdingbats standard font.

Definition at line 297 of file FontLibrary.cs.

7.135.3 Member Function Documentation

7.135.3.1 Add() [1/4]

```
void VectSharp.SimpleFontLibrary.Add (
            FontFamily fontFamily )
```

Add the specified font family to the library.

Parameters

fontFamily	The font family to add.

Definition at line 360 of file FontLibrary.cs.

7.135.3.2 Add() [2/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fileName )
```

Add the font family contained in the specified True Type Font file to the library.

Parameters

fileName	The path to the TTF file containing the font family.

Definition at line 372 of file FontLibrary.cs.

7.135.3.3 Add() [3/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fontFamily,
            string fileName )
```

Add the font family contained in the specified True Type Font file to the library, with the specified name. The font family is not loaded until it is requested for the first time.

Parameters

fontFamily	The name of the font family.
fileName	The path to the TTF file containing the font family.

Definition at line 387 of file FontLibrary.cs.

7.135.3.4 Add() [4/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fontFamilyName,
            FontFamily fontFamily )
```

Add the specified font family to the library with the specified name.

Parameters

fontFamilyName	The name of the font family.
fontFamily	The font family to add.

Definition at line 343 of file FontLibrary.cs.

7.135.3.5 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.SimpleFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 394 of file FontLibrary.cs.

7.135.3.6 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.SimpleFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 400 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.136 VectSharp.Size Struct Reference

Represents the size of an object.

Public Member Functions

- Size (double width, double height)

  Create a new Size.

Public Attributes

- double Width

  Width of the object.
- double Height

  Height of the object.

7.136.1 Detailed Description

Represents the size of an object.

Definition at line 268 of file Point.cs.

7.136.2 Constructor & Destructor Documentation

7.136.2.1 Size()

```
VectSharp.Size.Size (
            double width,
            double height )
```

Create a new Size.

Parameters

width	The width of the object.
height	The height of the object.

Definition at line 285 of file Point.cs.

7.136.3 Member Data Documentation

7.136.3.1 Height

```
double VectSharp.Size.Height
```

Height of the object.

Definition at line 278 of file Point.cs.

7.136.3.2 Width

```
double VectSharp.Size.Width
```

Width of the object.

Definition at line 273 of file Point.cs.

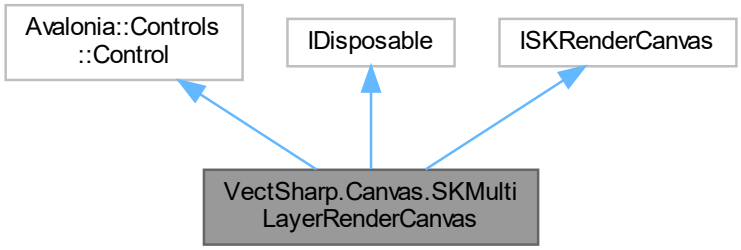
The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.137 VectSharp.Canvas.SKMultiLayerRenderCanvas Class Reference

Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.

Inheritance diagram for VectSharp.Canvas.SKMultiLayerRenderCanvas:



Public Member Functions

- SKMultiLayerRenderCanvas (Document document, Colour backgroundColour, double width, double height, List< SKRenderAction > layerTransforms=null)

  Create a new SKMultiLayerRenderCanvas from a Document, where each page represents a layer.
- SKMultiLayerRenderCanvas (IEnumerable< Page > layers, Colour backgroundColour, double width, double height, List< SKRenderAction > layerTransforms=null)

  Create a new SKMultiLayerRenderCanvas from a collection of Pages, each representing a layer.
- SKMultiLayerRenderCanvas (List< SKRenderContext > contents, List< SKRenderAction > contentTransforms, Colour backgroundColour, double width, double height)

  Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.
- void UpdateWith (List< SKRenderContext > contents, List< SKRenderAction > contentTransforms, Colour backgroundColour, double width, double height)

  Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.
- void UpdateLayer (int layer, SKRenderContext newContent, SKRenderAction newTransform)

  Replace a single layer with the specified content.
- void AddLayer (SKRenderContext newContent, SKRenderAction newTransform)

  Add a new layer to the image.
- void InsertLayer (int index, SKRenderContext newContent, SKRenderAction newTransform)

  Insert a new layer at the specified index.
- void RemoveLayer (int layer)

  Remove the specified layer from the image.
- void SwitchLayers (int layer1, int layer2)

  Switch the position of the two specified layers.
- void MoveLayer (int oldIndex, int newIndex)

  Move the specified layer to the specified position, shifting all other layers as necessary.
- RenderTargetBitmap RenderAtResolution (int width, int height, SKColor? background=null)

  Render the image to a bitmap at the specified resolution.
- void InvalidateDirty ()

  Invalidate the contents of the canvas, forcing it to redraw itself.
- void InvalidateZIndex ()

  Invalidate the contents of the canvas, specifying that the order of the layers has changed.
- override void Render (DrawingContext context)
- void Dispose ()

Public Attributes

- List< List< SKRenderAction > > RenderActions

  The list of render actions. Each element in this list is itself a list, containing the actions that correspond to a layer in the image.
- List< SKRenderAction > LayerTransforms

  The list of transforms associated with each layer.
- object RenderLock = new object()

  A lock for the rendering loop. The public methods of this class already lock on this, but you may need it if you want to directly manipulate the contents of the canvas.

Properties

- double PageWidth [get, set]

  The width of the page that is rendered on this canvas.
- double PageHeight [get, set]

  The height of the page that is rendered on this canvas.
- RelativePoint ClipMargin = new RelativePoint(0, 0, RelativeUnit.Absolute) [get, set]

  Defines an overdraw margin for the clipping area. Set this to a larger value if quickly moving the canvas around (e.g. in a ZoomBorder) causes the edge of the image to disappear for a few milliseconds and that annoys you. If RelativePoint.Unit is set to RelativeUnit.Absolute, the value corresponds to units on the VectSharp.Page; if it is set to RelativeUnit.Relative, it corresponds to a proportion of the visible area.

7.137.1 Detailed Description

Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.

Definition at line 39 of file SKMultiLayerRenderCanvas.cs.

7.137.2 Constructor & Destructor Documentation

7.137.2.1 SKMultiLayerRenderCanvas() [1/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            Document document,
            Colour backgroundColour,
            double width,
            double height,
            List< SKRenderAction > layerTransforms = null )
```

Create a new SKMultiLayerRenderCanvas from a Document, where each page represents a layer.

Parameters

document	The document containing the layers as Pages.
layerTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as the number of pages in *document* . This is useful to manipulate the position of each layer individually. If this is null, an identity transform is applied to each layer.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the pages it contains.
height	The height of the canvas and the pages it contains.

Definition at line 82 of file SKMultiLayerRenderCanvas.cs.

7.137.2.2 SKMultiLayerRenderCanvas() [2/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            IEnumerable< Page > layers,
            Colour backgroundColour,
            double width,
            double height,
            List< SKRenderAction > layerTransforms = null )
```

Create a new SKMultiLayerRenderCanvas from a collection of Pages, each representing a layer.

Parameters

layers	The contents of the canvas. Each element in this list represents a layer.
layerTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *layers* . This is useful to manipulate the position of each layer individually. If this is null, an identity transform is applied to each layer.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the pages it contains.
height	The height of the canvas and the pages it contains.

Definition at line 92 of file SKMultiLayerRenderCanvas.cs.

7.137.2.3 SKMultiLayerRenderCanvas() [3/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            List< SKRenderContext > contents,
            List< SKRenderAction > contentTransforms,
            Colour backgroundColour,
            double width,
            double height )
```

Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.

Parameters

contents	The contents of the canvas. Each element in this list represents a layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
contentTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *contents* . This is useful to manipulate the position of each layer individually.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the page it contains.
height	The height of the canvas and the page it contains.

Definition at line 125 of file SKMultiLayerRenderCanvas.cs.

7.137.3 Member Function Documentation

7.137.3.1 AddLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.AddLayer (
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Add a new layer to the image.

Parameters

newContent	The contents of the new layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
newTransform	The transform for the new layer.

Definition at line 256 of file SKMultiLayerRenderCanvas.cs.

7.137.3.2 Dispose()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.Dispose ( )
```

Definition at line 1134 of file SKMultiLayerRenderCanvas.cs.

7.137.3.3 InsertLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InsertLayer (
            int index,
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Insert a new layer at the specified index.

Parameters

index	The position at which the new layer will be inserted.
newContent	The contents of the new layer.
newTransform	The transform for the new layer.

Definition at line 289 of file SKMultiLayerRenderCanvas.cs.

7.137.3.4 InvalidateDirty()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InvalidateDirty ( )
```

Invalidate the contents of the canvas, forcing it to redraw itself.

Definition at line 943 of file SKMultiLayerRenderCanvas.cs.

7.137.3.5 InvalidateZIndex()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InvalidateZIndex ( )
```

Invalidate the contents of the canvas, specifying that the order of the layers has changed.

Definition at line 952 of file SKMultiLayerRenderCanvas.cs.

7.137.3.6 MoveLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.MoveLayer (
            int oldIndex,
            int newIndex )
```

Move the specified layer to the specified position, shifting all other layers as necessary.

Parameters

oldIndex	The current index of the layer to move.
newIndex	The final index of the layer. Layers after this will be shifted by 1 in order to accommodate the moved layer.

Definition at line 369 of file SKMultiLayerRenderCanvas.cs.

7.137.3.7 RemoveLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.RemoveLayer (
            int layer )
```

Remove the specified layer from the image.

Parameters

layer	The index of the layer to remove.

Definition at line 320 of file SKMultiLayerRenderCanvas.cs.

7.137.3.8 Render()

```
override void VectSharp.Canvas.SKMultiLayerRenderCanvas.Render (
            DrawingContext context )
```

Definition at line 977 of file SKMultiLayerRenderCanvas.cs.

7.137.3.9 RenderAtResolution()

```
RenderTargetBitmap VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderAtResolution (
            int width,
            int height,
            SKColor?  background = null )
```

Render the image to a bitmap at the specified resolution.

Parameters

width	The width of the rendered image. Note that the actual width of the returned image might be lower than this, depending on the aspect ratio of the image.
height	The height of the rendered image. Note that the actual height of the returned image might be lower than this, depending on the aspect ratio of the image.
background	The background colour for the image. If this is `null`, the current background colour is used.

Returns

A RenderTargetBitmap containing the image rendered at the specified resolution.

Definition at line 705 of file SKMultiLayerRenderCanvas.cs.

7.137.3.10 SwitchLayers()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.SwitchLayers (
            int layer1,
            int layer2 )
```

Switch the position of the two specified layers.

Parameters

layer1	The index of the first layer to switch.
layer2	The index of the second layer to switch.

Definition at line 344 of file SKMultiLayerRenderCanvas.cs.

7.137.3.11 UpdateLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.UpdateLayer (
            int layer,
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Replace a single layer with the specified content.

Parameters

layer	The index of the layer to replace.
newContent	The new contents of the layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
newTransform	The new transform for the layer.

Definition at line 224 of file SKMultiLayerRenderCanvas.cs.

7.137.3.12 UpdateWith()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.UpdateWith (
            List< SKRenderContext > contents,
            List< SKRenderAction > contentTransforms,
            Colour backgroundColour,
            double width,
            double height )
```

Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.

Parameters

contents	The contents of the canvas. Each element in this list represents a layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
contentTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *contents* . This is useful to manipulate the position of each layer individually.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the page it contains.
height	The height of the canvas and the page it contains.

Definition at line 146 of file SKMultiLayerRenderCanvas.cs.

7.137.4 Member Data Documentation

7.137.4.1 LayerTransforms

```
List<SKRenderAction> VectSharp.Canvas.SKMultiLayerRenderCanvas.LayerTransforms
```

The list of transforms associated with each layer.

Definition at line 69 of file SKMultiLayerRenderCanvas.cs.

7.137.4.2 RenderActions

```
List<List<SKRenderAction> > VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderActions
```

The list of render actions. Each element in this list is itself a list, containing the actions that correspond to a layer in the image.

Definition at line 64 of file SKMultiLayerRenderCanvas.cs.

7.137.4.3 RenderLock

```
object VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderLock = new object()
```

A lock for the rendering loop. The public methods of this class already lock on this, but you may need it if you want to directly manipulate the contents of the canvas.

Definition at line 696 of file SKMultiLayerRenderCanvas.cs.

7.137.5 Property Documentation

7.137.5.1 ClipMargin

```
RelativePoint VectSharp.Canvas.SKMultiLayerRenderCanvas.ClipMargin = new RelativePoint(0, 0,
RelativeUnit.Absolute)  [get], [set]
```

Defines an overdraw margin for the clipping area. Set this to a larger value if quickly moving the canvas around (e.g. in a ZoomBorder) causes the edge of the image to disappear for a few milliseconds and that annoys you. If RelativePoint.Unit is set to RelativeUnit.Absolute, the value corresponds to units on the VectSharp.Page; if it is set to RelativeUnit.Relative, it corresponds to a proportion of the visible area.

Definition at line 59 of file SKMultiLayerRenderCanvas.cs.

7.137.5.2 PageHeight

```
double VectSharp.Canvas.SKMultiLayerRenderCanvas.PageHeight  [get], [set]
```

The height of the page that is rendered on this canvas.

Definition at line 49 of file SKMultiLayerRenderCanvas.cs.

7.137.5.3 PageWidth

```
double VectSharp.Canvas.SKMultiLayerRenderCanvas.PageWidth  [get], [set]
```

The width of the page that is rendered on this canvas.

Definition at line 44 of file SKMultiLayerRenderCanvas.cs.

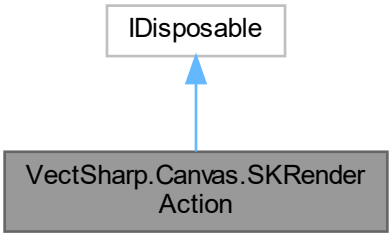
The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKMultiLayerRenderCanvas.cs

7.138 VectSharp.Canvas.SKRenderAction Class Reference

Represents a light-weight rendering action.

Inheritance diagram for VectSharp.Canvas.SKRenderAction:



Public Types

- enum ActionTypes

  Types of rendering actions.

Public Member Functions

- void InvalidateHitTestPath ()
- void InvalidateVisual ()
- void InvalidateZIndex ()
- void InvalidateAll ()
- void Dispose ()

Static Public Member Functions

- static SKRenderAction PathAction (SKPath path, SKPaint paint, string tag=null)

  Creates a new SKRenderAction representing a path.
- static SKRenderAction ClipAction (SKPath clippingPath, string tag=null)

  Creates a new SKRenderAction representing a clipping action.
- static SKRenderAction TextAction (string text, float x, float y, SKFont font, SKPaint paint, string tag=null)

  Creates a new SKRenderAction representing text.
- static SKRenderAction ImageAction (string imageId, SKRect sourceRect, SKRect destinationRect, string tag=null)

  Creates a new SKRenderAction representing an image.
- static SKRenderAction TransformAction (SKMatrix transform, string tag=null)

  Creates a new SKRenderAction representing a transform.
- static SKRenderAction SaveAction (string tag=null)

  Creates a new SKRenderAction that saves the current graphics state.
- static SKRenderAction RestoreAction (string tag=null)

  Creates a new SKRenderAction that saves the current graphics state.
- static SKRenderAction DrawFilteredGraphicsAction (SKRenderContext graphics, IFilter filter, string tag=null)

  Create a new SKRenderAction that draws some graphics with a filter.

Properties

- bool Disposed `[get]`

  Returns a boolean value indicating whether the current instance has been disposed.
- ActionTypes ActionType `[get]`

  Type of the rendering action.
- SKPath Path `[get, set]`

  Path that needs to be rendered (null if the action type is not ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- string Text `[get, set]`

  Text that needs to be rendered (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- SKFont Font `[get, set]`

  The font that will be used to render the text (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- float TextX `[get, set]`

  The X coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- float TextY `[get, set]`

  The Y coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- SKPaint Paint `[get, set]`

Paint used to render the text or path (`null` if the action type is neither ActionTypes.Text nor ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

- string ImageId `[get, set]`

  Univocal identifier of the image that needs to be drawn.

- SKRect? ImageSource `[get, set]`

  The source rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateVisual method.

- SKRect? ImageDestination `[get, set]`

  The destination rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateHitTestPath method.

- SKMatrix? Transform = null `[get, set]`

  The transformation matrix that will be applied to the current coordinate system (`null` if the action type is not ActionTypes.Transform). If you change this, you probably want to call this object's InvalidateVisual method.

- string Tag `[get]`

  A tag to access the SKRenderAction.

- uint ZIndex = 0 `[get, set]`

  The Z-index of the rendering action (an action with a higher Z-index will always appear above an action with a lower Z-index). The more different values there are for the Z-index, the slower the rendering, so keep use of this property to a minimum. If you change this, you probably want to call this object's InvalidateZIndex method.

- SKRenderContext Graphics = null `[get, set]`

  The graphics that will be drawn with the specified Filter. If you change this, you probably want to call this object's InvalidateVisual method.

- IFilter Filter `[get, set]`

  The filter with which the Graphics is drawn. If you change this, you probably want to call this object's InvalidateVisual method.

- object Payload `[get, set]`

  An arbitrary object associated with the RenderAction.

- Avalonia.Controls.Canvas Parent `[get]`

  The container of this SKRenderAction.

Events

- EventHandler< Avalonia.Input.PointerEventArgs > PointerEntered

  Raised when the pointer enters the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerEventArgs > PointerExited

  Raised when the pointer leaves the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerPressedEventArgs > PointerPressed

  Raised when the pointer is pressed while over the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerReleasedEventArgs > PointerReleased

  Raised when the pointer is released after a PointerPressed event.

7.138.1 Detailed Description

Represents a light-weight rendering action.

Definition at line 30 of file SKRenderContext.cs.

7.138.2 Member Enumeration Documentation

7.138.2.1 ActionTypes

```
enum VectSharp.Canvas.SKRenderAction.ActionTypes
```

Types of rendering actions.

Definition at line 42 of file SKRenderContext.cs.

7.138.3 Member Function Documentation

7.138.3.1 ClipAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.ClipAction (
            SKPath clippingPath,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a clipping action.

Parameters

clippingPath	The path to be used for clipping.
tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction representing a clipping action.

Definition at line 345 of file SKRenderContext.cs.

7.138.3.2 Dispose()

```
void VectSharp.Canvas.SKRenderAction.Dispose ( )
```

Definition at line 494 of file SKRenderContext.cs.

7.138.3.3 DrawFilteredGraphicsAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.DrawFilteredGraphicsAction (
            SKRenderContext graphics,
            IFilter filter,
            string tag = null )  [static]
```

Create a new SKRenderAction that draws some graphics with a filter.

Returns

A new SKRenderAction that draws some graphics with a filter.

Definition at line 464 of file SKRenderContext.cs.

7.138.3.4 ImageAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.ImageAction (
            string imageId,
            SKRect sourceRect,
            SKRect destinationRect,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing an image.

Parameters

imageId	The univocal identifier of the image to draw.
sourceRect	The source rectangle of the image.
destinationRect	The destination rectangle of the image.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing an image.

Definition at line 397 of file SKRenderContext.cs.

7.138.3.5 InvalidateAll()

```
void VectSharp.Canvas.SKRenderAction.InvalidateAll ( )
```

This methods signals to the Parent that the Z-index, shape and visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 303 of file SKRenderContext.cs.

7.138.3.6 InvalidateHitTestPath()

```
void VectSharp.Canvas.SKRenderAction.InvalidateHitTestPath ( )
```

Signals to this object that its shape has changed and a new path needs to be computed for the purpose of hit-testing. Also signals to the Parent that the visual properties of this object have changed and triggers a redraw.

This method should be called whenever the "shape" of the object represented by the SKRenderAction changes. If only the visual properties of this object have changed (e.g. the colour), call the InvalidateVisual method instead.

If you make changes to more than one SKRenderAction contained in the same SKMultiLayerRenderCanvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 232 of file SKRenderContext.cs.

7.138.3.7 InvalidateVisual()

```
void VectSharp.Canvas.SKRenderAction.InvalidateVisual ( )
```

This methods signals to the Parent that the visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If the "shape" of the object has changed as well, call the InvalidateHitTestPath method instead. If the Z-index of the object has changed, call the InvalidateZIndex method instead. If both the "shape" and the Z-index of the object have changed, call the InvalidateAll method.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 282 of file SKRenderContext.cs.

7.138.3.8 InvalidateZIndex()

```
void VectSharp.Canvas.SKRenderAction.InvalidateZIndex ( )
```

This methods signals to the Parent that the Z-index and visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If the "shape" of the object has changed as well, call the InvalidateAll method instead.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 293 of file SKRenderContext.cs.

7.138.3.9 PathAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.PathAction (
            SKPath path,
            SKPaint paint,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a path.

Parameters

path	The geometry to be rendered.
paint	The paint used to fill or stroke the path.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing a path.

Definition at line 321 of file SKRenderContext.cs.

7.138.3.10 RestoreAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.RestoreAction (
            string tag = null )  [static]
```

Creates a new SKRenderAction that saves the current graphics state.

Parameters

tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction that restores the last saved graphics state.

Definition at line 451 of file SKRenderContext.cs.

7.138.3.11 SaveAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.SaveAction (
            string tag = null )  [static]
```

Creates a new SKRenderAction that saves the current graphics state.

Parameters

tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction that saves the current graphics state.

Definition at line 437 of file SKRenderContext.cs.

7.138.3.12 TextAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.TextAction (
            string text,
```

```
            float x,
            float y,
            SKFont font,
            SKPaint paint,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing text.

Parameters

text	The text to be rendered.
x	The X coordinate at which the text will be drawn.
y	The Y coordinate at which the text will be drawn.
font	The font to be used to render the text.
paint	The paint to be used to fill or stroke the text.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing text.

Definition at line 365 of file SKRenderContext.cs.

7.138.3.13 TransformAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.TransformAction (
            SKMatrix transform,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a transform.

Parameters

transform	The transform to apply.
tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction representing a transform.

Definition at line 422 of file SKRenderContext.cs.

7.138.4 Property Documentation

7.138.4.1 ActionType

```
ActionTypes VectSharp.Canvas.SKRenderAction.ActionType [get]
```

Type of the rendering action.

Definition at line 88 of file SKRenderContext.cs.

7.138.4.2 Disposed

```
bool VectSharp.Canvas.SKRenderAction.Disposed [get]
```

Returns a boolean value indicating whether the current instance has been disposed.

Definition at line 35 of file SKRenderContext.cs.

7.138.4.3 Filter

```
IFilter VectSharp.Canvas.SKRenderAction.Filter [get], [set]
```

The filter with which the Graphics is drawn. If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 163 of file SKRenderContext.cs.

7.138.4.4 Font

```
SKFont VectSharp.Canvas.SKRenderAction.Font [get], [set]
```

The font that will be used to render the text (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 106 of file SKRenderContext.cs.

7.138.4.5 Graphics

```
SKRenderContext VectSharp.Canvas.SKRenderAction.Graphics = null [get], [set]
```

The graphics that will be drawn with the specified Filter. If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 158 of file SKRenderContext.cs.

7.138.4.6 ImageDestination

```
SKRect?  VectSharp.Canvas.SKRenderAction.ImageDestination  [get], [set]
```

The destination rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 136 of file SKRenderContext.cs.

7.138.4.7 ImageId

```
string VectSharp.Canvas.SKRenderAction.ImageId  [get], [set]
```

Univocal identifier of the image that needs to be drawn.

Definition at line 126 of file SKRenderContext.cs.

7.138.4.8 ImageSource

```
SKRect?  VectSharp.Canvas.SKRenderAction.ImageSource  [get], [set]
```

The source rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 131 of file SKRenderContext.cs.

7.138.4.9 Paint

```
SKPaint VectSharp.Canvas.SKRenderAction.Paint  [get], [set]
```

Paint used to render the text or path (`null` if the action type is neither ActionTypes.Text nor ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 121 of file SKRenderContext.cs.

7.138.4.10 Parent

```
Avalonia.Controls.Canvas VectSharp.Canvas.SKRenderAction.Parent  [get]
```

The container of this SKRenderAction.

Definition at line 175 of file SKRenderContext.cs.

7.138.4.11 Path

```
SKPath VectSharp.Canvas.SKRenderAction.Path  [get], [set]
```

Path that needs to be rendered (null if the action type is not ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 93 of file SKRenderContext.cs.

7.138.4.12 Payload

```
object VectSharp.Canvas.SKRenderAction.Payload  [get], [set]
```

An arbitrary object associated with the RenderAction.

Definition at line 168 of file SKRenderContext.cs.

7.138.4.13 Tag

```
string VectSharp.Canvas.SKRenderAction.Tag  [get]
```

A tag to access the SKRenderAction.

Definition at line 146 of file SKRenderContext.cs.

7.138.4.14 Text

```
string VectSharp.Canvas.SKRenderAction.Text  [get], [set]
```

Text that needs to be rendered (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 101 of file SKRenderContext.cs.

7.138.4.15 TextX

```
float VectSharp.Canvas.SKRenderAction.TextX  [get], [set]
```

The X coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 111 of file SKRenderContext.cs.

7.138.4.16 TextY

```
float VectSharp.Canvas.SKRenderAction.TextY  [get], [set]
```

The Y coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 116 of file SKRenderContext.cs.

7.138.4.17 Transform

```
SKMatrix?  VectSharp.Canvas.SKRenderAction.Transform = null  [get], [set]
```

The transformation matrix that will be applied to the current coordinate system (`null` if the action type is not ActionTypes.Transform). If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 141 of file SKRenderContext.cs.

7.138.4.18 ZIndex

```
uint VectSharp.Canvas.SKRenderAction.ZIndex = 0  [get], [set]
```

The Z-index of the rendering action (an action with a higher Z-index will always appear above an action with a lower Z-index). The more different values there are for the Z-index, the slower the rendering, so keep use of this property to a minimum. If you change this, you probably want to call this object's InvalidateZIndex method.

Definition at line 153 of file SKRenderContext.cs.

7.138.5 Event Documentation

7.138.5.1 PointerEntered

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.SKRenderAction.PointerEntered
```

Raised when the pointer enters the area covered by the SKRenderAction.

Definition at line 186 of file SKRenderContext.cs.

7.138.5.2 PointerExited

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.SKRenderAction.PointerExited
```

Raised when the pointer leaves the area covered by the SKRenderAction.

Definition at line 191 of file SKRenderContext.cs.

7.138.5.3 PointerPressed

```
EventHandler<Avalonia.Input.PointerPressedEventArgs> VectSharp.Canvas.SKRenderAction.Pointer↩
Pressed
```

Raised when the pointer is pressed while over the area covered by the SKRenderAction.

Definition at line 196 of file SKRenderContext.cs.

7.138.5.4 PointerReleased

```
EventHandler<Avalonia.Input.PointerReleasedEventArgs> VectSharp.Canvas.SKRenderAction.Pointer↩
Released
```

Raised when the pointer is released after a PointerPressed event.

Definition at line 201 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.139 VectSharp.Canvas.SKRenderContext Class Reference

Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.

7.139.1 Detailed Description

Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.

Definition at line 546 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.140 VectSharp.Canvas.SKRenderContextInterpreter Class Reference

Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Static Public Member Functions

- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.
- static SKRenderContext CopyToSKRenderContext (this Page page, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static SKRenderContext CopyToSKRenderContext (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static SKRenderContext CopyToSKRenderContext (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.

7.140.1 Detailed Description

Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Definition at line 1545 of file SKRenderContext.cs.

7.140.2 Member Function Documentation

7.140.2.1 CopyToSKRenderContext() [1/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1678 of file SKRenderContext.cs.

7.140.2.2 CopyToSKRenderContext() [2/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1695 of file SKRenderContext.cs.

7.140.2.3 CopyToSKRenderContext() [3/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.

Parameters

removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1714 of file SKRenderContext.cs.

7.140.2.4 PaintToSKCanvas() [1/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1592 of file SKRenderContext.cs.

7.140.2.5 PaintToSKCanvas() [2/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
            bool removeTaggedActionsAfterExecution = true,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1614 of file SKRenderContext.cs.

7.140.2.6 PaintToSKCanvas() [3/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1572 of file SKRenderContext.cs.

7.140.2.7 PaintToSKCanvas() [4/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1628 of file SKRenderContext.cs.

7.140.2.8 PaintToSKCanvas() [5/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1645 of file SKRenderContext.cs.

7.140.2.9 PaintToSKCanvas() [6/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
```

```
bool removeTaggedActionsAfterExecution = true,
AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1664 of file SKRenderContext.cs.

7.140.2.10 Rasterise()

```
static RasterImage VectSharp.Canvas.SKRenderContextInterpreter.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 1736 of file SKRenderContext.cs.

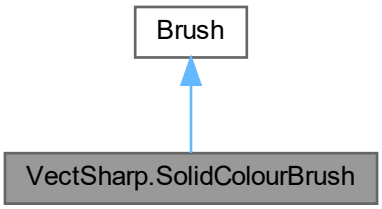
The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.141 VectSharp.SolidColourBrush Class Reference

Represents a brush painting with a single solid colour.

Inheritance diagram for VectSharp.SolidColourBrush:



Public Member Functions

- SolidColourBrush (Colour colour)

  Creates a new SolidColourBrush with the specified colour .
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Static Public Member Functions

- static implicit operator SolidColourBrush (Colour colour)

  Implicitly converts a Colour into a SolidColourBrush.

Properties

- Colour Colour [get]

  The colour of the brush.
- double R [get]

  Red component of the colour. Range: [0, 1].
- double G [get]

  Green component of the colour. Range: [0, 1].
- double B [get]

  Blue component of the colour. Range: [0, 1].
- double A [get]

  Alpha component of the colour. Range: [0, 1].

7.141.1 Detailed Description

Represents a brush painting with a single solid colour.

Definition at line 54 of file Brush.cs.

7.141.2 Constructor & Destructor Documentation

7.141.2.1 SolidColourBrush()

```
VectSharp.SolidColourBrush.SolidColourBrush (
            Colour colour )
```

Creates a new SolidColourBrush with the specified *colour* .

Parameters

colour	The Colour to use for the brush.

Definition at line 85 of file Brush.cs.

7.141.3 Member Function Documentation

7.141.3.1 MultiplyOpacity()

```
override Brush VectSharp.SolidColourBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 91 of file Brush.cs.

7.141.3.2 operator SolidColourBrush()

```
static implicit VectSharp.SolidColourBrush.operator SolidColourBrush (
            Colour colour )  [static]
```

Implicitly converts a Colour into a SolidColourBrush.

Parameters

colour	The Colour to use for the brush.

Definition at line 100 of file Brush.cs.

7.141.4 Property Documentation

7.141.4.1 A

```
double VectSharp.SolidColourBrush.A  [get]
```

Alpha component of the colour. Range: [0, 1].

Definition at line 79 of file Brush.cs.

7.141.4.2 B

```
double VectSharp.SolidColourBrush.B  [get]
```

Blue component of the colour. Range: [0, 1].

Definition at line 74 of file Brush.cs.

7.141.4.3 Colour

```
Colour VectSharp.SolidColourBrush.Colour  [get]
```

The colour of the brush.

Definition at line 59 of file Brush.cs.

7.141.4.4 G

```
double VectSharp.SolidColourBrush.G  [get]
```

Green component of the colour. Range: [0, 1].

Definition at line 69 of file Brush.cs.

7.141.4.5 R

```
double VectSharp.SolidColourBrush.R  [get]
```

Red component of the colour. Range: [0, 1].

Definition at line 64 of file Brush.cs.

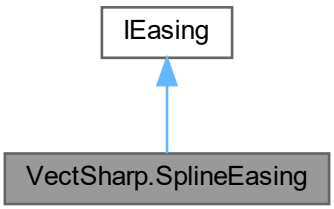
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.142 VectSharp.SplineEasing Class Reference

Describes an easing defined by a Cubic Bezier curve.

Inheritance diagram for VectSharp.SplineEasing:



Public Member Functions

- SplineEasing (Point controlPoint1, Point controlPoint2)

  Creates a new SplineEasing with the specified control points. The start point is always (0, 0) and the end point is always (1, 1).
- double Ease (double value)

  Applies the easing to the specified transition offset.

Parameters

value	*The transition offset (ranging from 0 to 1).*

Returns

The eased transition offset value.

Properties

- Point ControlPoint1 `[get]`

  The first control point of the curve.
- Point ControlPoint2 `[get]`

  The second control point of the curve.

7.142.1 Detailed Description

Describes an easing defined by a Cubic Bezier curve.

Definition at line 1233 of file Animation.cs.

7.142.2 Constructor & Destructor Documentation

7.142.2.1 SplineEasing()

```
VectSharp.SplineEasing.SplineEasing (
            Point controlPoint1,
            Point controlPoint2 )
```

Creates a new SplineEasing with the specified control points. The start point is always (0, 0) and the end point is always (1, 1).

Parameters

controlPoint1	The first control point of the curve. Both X and Y must be between 0 and 1, inclusive.
controlPoint2	The second control point of the curve. Both X and Y must be between 0 and 1, inclusive.

Exceptions

ArgumentException	This exception is thrown if any coordinate of the control points is < 0 or > 1.

Definition at line 1252 of file Animation.cs.

7.142.3 Member Function Documentation

7.142.3.1 Ease()

```
double VectSharp.SplineEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implements VectSharp.IEasing.

Definition at line 1301 of file Animation.cs.

7.142.4 Property Documentation

7.142.4.1 ControlPoint1

```
Point VectSharp.SplineEasing.ControlPoint1  [get]
```

The first control point of the curve.

Definition at line 1238 of file Animation.cs.

7.142.4.2 ControlPoint2

```
Point VectSharp.SplineEasing.ControlPoint2  [get]
```

The second control point of the curve.

Definition at line 1243 of file Animation.cs.

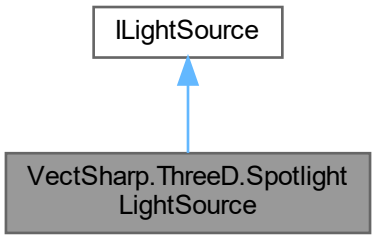
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.143 VectSharp.ThreeD.SpotlightLightSource Class Reference

Represents a conic spotlight.

Inheritance diagram for VectSharp.ThreeD.SpotlightLightSource:



Public Member Functions

- SpotlightLightSource (double intensity, Point3D position, NormalizedVector3D direction, double beamWidth↩ Angle, double cutoffAngle)

  Creates a new SpotlightLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get, set]`

The position of the light source.

- NormalizedVector3D Direction `[get, set]`

  The direction of the cone axis.

- double Intensity `[get, set]`

  The base intensity of the light.

- double BeamWidthAngle `[get, set]`

  The angular size of the light cone, in radians.

- double CutoffAngle `[get, set]`

  The angular size of the cutoff cone, in radians.

- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

- double AngleAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates between the main light cone and the cutoff cone.

7.143.1 Detailed Description

Represents a conic spotlight.

Definition at line 256 of file Lights.cs.

7.143.2 Constructor & Destructor Documentation

7.143.2.1 SpotlightLightSource()

```
VectSharp.ThreeD.SpotlightLightSource.SpotlightLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double beamWidthAngle,
            double cutoffAngle )
```

Creates a new SpotlightLightSource instance.

Parameters

intensity	The intensity of the light.
position	The position of the light source.
direction	The direction of the cone's axis.
beamWidthAngle	The angular size of the light cone, in radians.
cutoffAngle	The angular size of the cutoff cone, in radians.

Definition at line 304 of file Lights.cs.

7.143.3 Member Function Documentation

7.143.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.SpotlightLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 314 of file Lights.cs.

7.143.3.2 GetObstruction()

```
double VectSharp.ThreeD.SpotlightLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 362 of file Lights.cs.

7.143.4 Property Documentation

7.143.4.1 AngleAttenuationExponent

```
double VectSharp.ThreeD.SpotlightLightSource.AngleAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates between the main light cone and the cutoff cone.

Definition at line 294 of file Lights.cs.

7.143.4.2 BeamWidthAngle

```
double VectSharp.ThreeD.SpotlightLightSource.BeamWidthAngle  [get], [set]
```

The angular size of the light cone, in radians.

Definition at line 279 of file Lights.cs.

7.143.4.3 CastsShadow

```
bool VectSharp.ThreeD.SpotlightLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 259 of file Lights.cs.

7.143.4.4 CutoffAngle

```
double VectSharp.ThreeD.SpotlightLightSource.CutoffAngle  [get], [set]
```

The angular size of the cutoff cone, in radians.

Definition at line 284 of file Lights.cs.

7.143.4.5 Direction

```
NormalizedVector3D VectSharp.ThreeD.SpotlightLightSource.Direction  [get], [set]
```

The direction of the cone axis.

Definition at line 269 of file Lights.cs.

7.143.4.6 DistanceAttenuationExponent

```
double VectSharp.ThreeD.SpotlightLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 289 of file Lights.cs.

7.143.4.7 Intensity

```
double VectSharp.ThreeD.SpotlightLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 274 of file Lights.cs.

7.143.4.8 Position

```
Point3D VectSharp.ThreeD.SpotlightLightSource.Position  [get], [set]
```

The position of the light source.

Definition at line 264 of file Lights.cs.

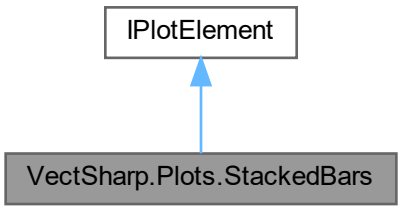
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.144 VectSharp.Plots.StackedBars Class Reference

A plot element that draws stacked bars.

Inheritance diagram for VectSharp.Plots.StackedBars:



Public Member Functions

- StackedBars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new StackedBars instance.
- StackedBars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new StackedBars instance.
- StackedBars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new StackedBars instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- bool Vertical = true `[get, set]`

  If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.
- SortedSet< IReadOnlyList< double > > Data `[get, set]`

  The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
- Func< IReadOnlyList< double >, IReadOnlyList< double > > GetBaseline `[get, set]`

  A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
- double Margin `[get, set]`

  The margin between consecutive bars. This should range between 0 and 1.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } `[get, set]`

  Presentation attributes for the bars. An element from this collection is used for each segment in the stack; if there are more segments than elements in this collection, the presentation attributes are wrapped.
- string Tag `[get, set]`

  A tag to identify the stacked bars in the plot.

7.144.1 Detailed Description

A plot element that draws stacked bars.

Definition at line 325 of file Bars.cs.

7.144.2 Constructor & Destructor Documentation

7.144.2.1 StackedBars() [1/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 399 of file Bars.cs.

7.144.2.2 StackedBars() [2/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 418 of file Bars.cs.

7.144.2.3 StackedBars() [3/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if *vertical* is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 430 of file Bars.cs.

7.144.3 Member Function Documentation

7.144.3.1 Plot()

```
void VectSharp.Plots.StackedBars.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 433 of file Bars.cs.

7.144.4 Property Documentation

7.144.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.StackedBars.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 372 of file Bars.cs.

7.144.4.2 Data

```
SortedSet<IReadOnlyList<double> > VectSharp.Plots.StackedBars.Data  [get], [set]
```

The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.

Definition at line 341 of file Bars.cs.

7.144.4.3 GetBaseline

```
Func<IReadOnlyList<double>, IReadOnlyList<double> > VectSharp.Plots.StackedBars.GetBaseline
[get], [set]
```

A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnly↩ List<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.

Definition at line 348 of file Bars.cs.

7.144.4.4 Margin

```
double VectSharp.Plots.StackedBars.Margin  [get], [set]
```

The margin between consecutive bars. This should range between 0 and 1.

Definition at line 353 of file Bars.cs.

7.144.4.5 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.StackedBars.Presentation↩
Attributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes()
}  [get], [set]
```

Presentation attributes for the bars. An element from this collection is used for each segment in the stack; if there are more segments than elements in this collection, the presentation attributes are wrapped.

Definition at line 380 of file Bars.cs.

7.144.4.6 Tag

```
string VectSharp.Plots.StackedBars.Tag  [get], [set]
```

A tag to identify the stacked bars in the plot.

Definition at line 385 of file Bars.cs.

7.144.4.7 Vertical

```
bool VectSharp.Plots.StackedBars.Vertical = true  [get], [set]
```

If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 333 of file Bars.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.145 VectSharp.SVG.SVGContextInterpreter Class Reference

Contains methods to render a Page as an SVG file.

Classes

- class FilterOption
  Determines how and whether image filters are rasterised.

Public Types

- enum TextOptions
  Defines whether the used fonts should be included in the file.

Static Public Member Functions

- static void SaveAsSVG (this Page page, string fileName, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG file.
- static void SaveAsSVG (this Page page, Stream stream, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG stream.
- static XmlDocument SaveAsSVG (this Page page, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG document.
- static XmlDocument SaveAsAnimatedSVG (this Animation animation, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG document, using SVG animations.
- static void SaveAsAnimatedSVG (this Animation animation, Stream stream, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG stream, using SVG animations.
- static void SaveAsAnimatedSVG (this Animation animation, string fileName, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG file, using SVG animations.
- static XmlDocument SaveAsAnimatedSVGWithFrames (this Animation animation, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG document, encoding discrete frames.
- static void SaveAsAnimatedSVGWithFrames (this Animation animation, Stream stream, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG stream, encoding discrete frames.
- static void SaveAsAnimatedSVGWithFrames (this Animation animation, string fileName, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG file, encoding discrete frames.

7.145.1 Detailed Description

Contains methods to render a Page as an SVG file.

Definition at line 2226 of file SVGContext.cs.

7.145.2 Member Enumeration Documentation

7.145.2.1 TextOptions

`enum VectSharp.SVG.SVGContextInterpreter.TextOptions`

Defines whether the used fonts should be included in the file.

Definition at line 2249 of file SVGContext.cs.

7.145.3 Member Function Documentation

7.145.3.1 SaveAsAnimatedSVG() [1/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG document, using SVG animations.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An XmlDocument containing the animated SVG image.

Definition at line 2719 of file SVGContext.cs.

7.145.3.2 SaveAsAnimatedSVG() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
            Stream stream,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG stream, using SVG animations.

Parameters

animation	The Animation to render.
stream	The Stream on which the SVG document will be written.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3218 of file SVGContext.cs.

7.145.3.3 SaveAsAnimatedSVG() [3/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
            string fileName,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG file, using SVG animations.

Parameters

animation	The Animation to render.
fileName	The output file that will be created.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3234 of file SVGContext.cs.

7.145.3.4 SaveAsAnimatedSVGWithFrames() [1/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG document, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An XmlDocument containing the animated SVG image.

Definition at line 3253 of file SVGContext.cs.

7.145.3.5 SaveAsAnimatedSVGWithFrames() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            Stream stream,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG stream, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
stream	The Stream on which the SVG document will be written.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3333 of file SVGContext.cs.

7.145.3.6 SaveAsAnimatedSVGWithFrames() [3/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            string fileName,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG file, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
fileName	The output file that will be created.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3350 of file SVGContext.cs.

7.145.3.7 SaveAsSVG() [1/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            Stream stream,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG stream.

Parameters

page	The Page to render.
stream	The stream to which the SVG data will be written.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Definition at line 2361 of file SVGContext.cs.

7.145.3.8 SaveAsSVG() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            string fileName,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG file.

Parameters

page	The Page to render.
fileName	The full path to the file to save. If it exists, it will be overwritten.
textOption	Defines whether the used fonts should be included in the file.

Parameters

linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Definition at line 2238 of file SVGContext.cs.

7.145.3.9 SaveAsSVG() [3/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG document.

Parameters

page	The Page to render.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Returns

An XmlDocument containing the rendered SVG image.

Definition at line 2376 of file SVGContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGContext.cs

7.146 VectSharp.Markdown.SyntaxHighlighter Class Reference

Contains methods to perform syntax highlighting.

Static Public Member Functions

- static List< List< FormattedString > > GetSyntaxHighlightedLines (string sourceCode, string language)

  Performs syntax highlighting for a specified language on some source code.

7.146.1 Detailed Description

Contains methods to perform syntax highlighting.

Definition at line 73 of file SyntaxHighlighting.cs.

7.146.2 Member Function Documentation

7.146.2.1 GetSyntaxHighlightedLines()

```
static List< List< FormattedString > > VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlighted↩
Lines (
            string sourceCode,
            string language )  [static]
```

Performs syntax highlighting for a specified language on some source code.

Parameters

sourceCode	The source code to be highlighted.
language	The name of the language to use for the highlighting.

Returns

A list of lists of FormattedStrings. Each list of FormattedStrings represents a line.

Definition at line 129 of file SyntaxHighlighting.cs.

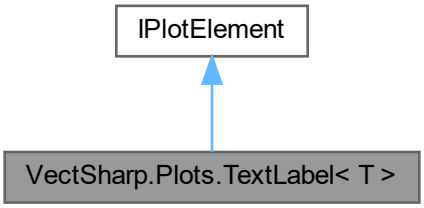
The documentation for this class was generated from the following file:

- VectSharp.Markdown/SyntaxHighlighting.cs

7.147 VectSharp.Plots.TextLabel< T > Class Template Reference

A plot element that draws a single text label.

Inheritance diagram for VectSharp.Plots.TextLabel< T >:



Public Member Functions

- TextLabel (string label, T position, ICoordinateSystem< T > coordinateSystem)

  Create a new TextLabel< T > instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- T Position `[get, set]`

  The position of the label, in data space coordinates.
- string Label `[get, set]`

  The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.Format(string, Font, Font, Font, F method).
- double Rotation = 0 `[get, set]`

  The angle at which the text is drawn, with respect to the horizontal.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the text.
- TextAnchors Alignment = TextAnchors.Center `[get, set]`

  The alignment for the text.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = null } `[get, set]`

  Presentation attributes determining the appearance (stroke and fill colour, etc.) of the text label.
- string Tag `[get, set]`

  A tag to identify the label in the plot.

7.147.1 Detailed Description

A plot element that draws a single text label.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 226 of file DataPoints.cs.

7.147.2 Constructor & Destructor Documentation

7.147.2.1 TextLabel()

```
VectSharp.Plots.TextLabel< T >.TextLabel (
            string label,
            T position,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new TextLabel<T> instance.

Parameters

label	The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.Format(string, Font, Font, Font, Font, Brush) method).
position	The position of the label, in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 275 of file DataPoints.cs.

7.147.3 Member Function Documentation

7.147.3.1 Plot()

```
void VectSharp.Plots.TextLabel< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 283 of file DataPoints.cs.

7.147.4 Property Documentation

7.147.4.1 Alignment

```
TextAnchors VectSharp.Plots.TextLabel< T >.Alignment = TextAnchors.Center  [get], [set]
```

The alignment for the text.

Definition at line 251 of file DataPoints.cs.

7.147.4.2 Baseline

```
TextBaselines VectSharp.Plots.TextLabel< T >.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the text.

Definition at line 246 of file DataPoints.cs.

7.147.4.3 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.TextLabel< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 256 of file DataPoints.cs.

7.147.4.4 Label

```
string VectSharp.Plots.TextLabel< T >.Label  [get], [set]
```

The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.Format(string, Font, Font, For
method).

Definition at line 236 of file DataPoints.cs.

7.147.4.5 Position

```
T VectSharp.Plots.TextLabel< T >.Position  [get], [set]
```

The position of the label, in data space coordinates.

Definition at line 231 of file DataPoints.cs.

7.147.4.6 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.TextLabel< T >.PresentationAttributes = new
PlotElementPresentationAttributes() { Stroke = null }  [get], [set]
```

Presentation attributes determining the appearance (stroke and fill colour, etc.) of the text label.

Definition at line 262 of file DataPoints.cs.

7.147.4.7 Rotation

```
double VectSharp.Plots.TextLabel< T >.Rotation = 0  [get], [set]
```

The angle at which the text is drawn, with respect to the horizontal.

Definition at line 241 of file DataPoints.cs.

7.147.4.8 Tag

```
string VectSharp.Plots.TextLabel< T >.Tag  [get], [set]
```

A tag to identify the label in the plot.

Definition at line 267 of file DataPoints.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.148 VectSharp.Transition Class Reference

Describes the transition between two successive Frames.

Public Member Functions

- Transition (double duration, IEasing easing=null, Dictionary< string, IEasing > easings=null)

  Creates a new Transition with the specified duration and easings.

Properties

- double Duration [get]

  The duration of the transition, in milliseconds.
- IEasing OverallEasing = null [get]

  The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.
- Dictionary< string, IEasing > Easings = null [get]

  A dictionary associating graphic action tags to the corresponding IEasing.

7.148.1 Detailed Description

Describes the transition between two successive Frames.

Definition at line 1341 of file Animation.cs.

7.148.2 Constructor & Destructor Documentation

7.148.2.1 Transition()

```
VectSharp.Transition.Transition (
            double duration,
            IEasing easing = null,
            Dictionary< string, IEasing > easings = null )
```

Creates a new Transition with the specified duration and easings.

Parameters

duration	The duration of the transition, in milliseconds.
easing	The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.
easings	A dictionary associating graphic action tags to the corresponding IEasing.

Definition at line 1364 of file Animation.cs.

7.148.3 Property Documentation

7.148.3.1 Duration

```
double VectSharp.Transition.Duration  [get]
```

The duration of the transition, in milliseconds.

Definition at line 1346 of file Animation.cs.

7.148.3.2 Easings

```
Dictionary<string, IEasing> VectSharp.Transition.Easings = null  [get]
```

A dictionary associating graphic action tags to the corresponding IEasing.

Definition at line 1356 of file Animation.cs.

7.148.3.3 OverallEasing

```
IEasing VectSharp.Transition.OverallEasing = null  [get]
```

The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.

Definition at line 1351 of file Animation.cs.

The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.149 VectSharp.TrueTypeFile Class Reference

Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.↩microsoft.com/en-us/typography/opentype/spec/

Classes

- struct Bearings

  Represents the left- and right-side bearings of a glyph.
- class PairKerning

  Contains information describing how the position of two glyphs in a kerning pair should be altered.
- struct TrueTypeName

  Represets a TrueType name.
- struct TrueTypePoint

  Represents a point in a TrueType path description.
- struct VerticalMetrics

  Represents the maximum heigth above and depth below the baseline of a glyph.

Public Member Functions

- void Destroy ()

  Remove this TrueType file from the cache, clear the tables and release the FontStream. Only call this when the actual file that was used to create this object needs to be changed!
- TrueTypeFile SubsetFont (string charactersToInclude, bool consolidateAt32=false, Dictionary< char, char > outputEncoding=null)

  Create a subset of the TrueType file, containing only the glyphs for the specified characters.
- string GetFontFamilyName ()

  Obtains the font family name from the TrueType file.
- string GetFullFontFamilyName ()

  Obtains the full font family name from the TrueType file.
- string GetFontName ()

  Obtains the PostScript font name from the TrueType file.
- ushort GetFirstCharIndex ()

  Returns the index of the first character glyph represented by the font.
- ushort GetLastCharIndex ()

  Returns the index of the last character glyph represented by the font.
- bool IsItalic ()

  Determines whether the typeface is Italic or Oblique or not.
- bool IsOblique ()

  Determines whether the typeface is Oblique or not.
- bool IsBold ()

  Determines whether the typeface is Bold or not.
- bool IsFixedPitch ()

  Determines whether the typeface is fixed-pitch (aka monospaces) or not.
- bool IsSerif ()

  Determines whether the typeface is serifed or not.
- bool IsScript ()

  Determines whether the typeface is a script typeface or not.
- int GetGlyphIndex (char glyph)

  Determines the index of the glyph corresponding to a certain character.
- TrueTypePoint[ ][ ] GetGlyphPath (int glyphIndex, double size)

  Get the path that describes the shape of a glyph.
- TrueTypePoint[ ][ ] GetGlyphPath (char glyph, double size)

  Get the path that describes the shape of a glyph.
- double Get1000EmGlyphWidth (char glyph)

  Computes the advance width of a glyph, in thousandths of em unit.
- double Get1000EmGlyphWidth (int glyphIndex)

  Computes the advance width of a glyph, in thousandths of em unit.
- double Get1000EmWinAscent ()

  Computes the font's Win ascent, in thousandths of em unit.
- double Get1000EmAscent ()

  Computes the font ascent, in thousandths of em unit.
- double Get1000EmDescent ()

  Computes the font descent, in thousandths of em unit.
- double Get1000EmYMax ()

  Computes the maximum height over the baseline of the font, in thousandths of em unit.
- double Get1000EmYMin ()

  Computes the maximum depth below the baseline of the font, in thousandths of em unit.
- double Get1000EmXMax ()

Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em unit.

- double Get1000EmXMin ()

  Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em unit.
- Bearings Get1000EmGlyphBearings (char glyph)

  Computes the left- and right- side bearings of a glyph, in thousandths of em unit.
- VerticalMetrics Get1000EmGlyphVerticalMetrics (char glyph)

  Computes the vertical metrics of a glyph, in thousandths of em unit.
- double Get1000EmUnderlinePosition ()

  Computes the distance of the top of the underline from the baseline, in thousandths of em unit.
- double Get1000EmUnderlineThickness ()

  Computes the thickness of the underline, in thousandths of em unit.
- int GetUnitsPerEm ()

  Returns the number of units for each em in the font file. Multiplying any glyph measurement by this size should generally lead to an integer value.
- double GetItalicAngle ()

  Computes the italic angle for the current font, in thousandths of em unit. This is computed from the vertical and is negative for text that leans forwards.
- double[ ] Get1000EmUnderlineIntersections (char glyph, double position, double thickness)

  Computes the intersections between an underline at the specified position and thickness and a glyph, in thousandths of em units.
- PairKerning Get1000EmKerning (char glyph1, char glyph2)

  Gets the kerning between two glyphs.
- PairKerning Get1000EmKerning (int glyph1Index, int glyph2Index)

  Gets the kerning between two glyphs.

Static Public Member Functions

- static bool GetNames (string file, out List< TrueTypeName > names)

  Determines whether the file is a valid TrueType file and retrieves the list of names defined in it.
- static bool GetNames (Stream fs, out List< TrueTypeName > names)

  Determines whether the stream contains a valid TrueType file and retrieves the list of names defined in it.

Properties

- Stream FontStream `[get]`

  A stream pointing to the TrueType file source (either on disk or in memory). Never dispose this stream directly; if you really need to, call Destroy instead.

7.149.1 Detailed Description

Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.↩microsoft.com/en-us/typography/opentype/spec/

Definition at line 29 of file TrueType.cs.

7.149.2 Member Function Documentation

7.149.2.1 Destroy()

```
void VectSharp.TrueTypeFile.Destroy ( )
```

Remove this TrueType file from the cache, clear the tables and release the FontStream. Only call this when the actual file that was used to create this object needs to be changed!

Definition at line 52 of file TrueType.cs.

7.149.2.2 Get1000EmAscent()

```
double VectSharp.TrueTypeFile.Get1000EmAscent ( )
```

Computes the font ascent, in thousandths of em unit.

Returns

The font ascent in thousandths of em unit.

Definition at line 2265 of file TrueType.cs.

7.149.2.3 Get1000EmDescent()

```
double VectSharp.TrueTypeFile.Get1000EmDescent ( )
```

Computes the font descent, in thousandths of em unit.

Returns

The font descent in thousandths of em unit.

Definition at line 2275 of file TrueType.cs.

7.149.2.4 Get1000EmGlyphBearings()

```
Bearings VectSharp.TrueTypeFile.Get1000EmGlyphBearings (
            char glyph )
```

Computes the left- and right- side bearings of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose bearings are to be computed.

Returns

The left- and right- side bearings of the glyph in thousandths of em unit

Definition at line 2357 of file TrueType.cs.

7.149.2.5 Get1000EmGlyphVerticalMetrics()

```
VerticalMetrics VectSharp.TrueTypeFile.Get1000EmGlyphVerticalMetrics (
            char glyph )
```

Computes the vertical metrics of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose vertical metrics are to be computed.

Returns

The vertical metrics of a glyph, in thousandths of em unit.

Definition at line 2405 of file TrueType.cs.

7.149.2.6 Get1000EmGlyphWidth() [1/2]

```
double VectSharp.TrueTypeFile.Get1000EmGlyphWidth (
            char glyph )
```

Computes the advance width of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose advance width is to be computed.

Returns

The advance width of the glyph in thousandths of em unit.

Definition at line 2216 of file TrueType.cs.

7.149.2.7 Get1000EmGlyphWidth() [2/2]

```
double VectSharp.TrueTypeFile.Get1000EmGlyphWidth (
            int glyphIndex )
```

Computes the advance width of a glyph, in thousandths of em unit.

Parameters

glyphIndex	The index of the glyph whose advance width is to be computed.

Returns

The advance width of the glyph in thousandths of em unit.

Definition at line 2234 of file TrueType.cs.

7.149.2.8 Get1000EmKerning() [1/2]

```
PairKerning VectSharp.TrueTypeFile.Get1000EmKerning (
            char glyph1,
            char glyph2 )
```

Gets the kerning between two glyphs.

Parameters

glyph1	The first glyph of the kerning pair.
glyph2	The second glyph of the kerning pair.

Returns

A PairKerning object containing information about how the position of each glyphs should be altered.

Definition at line 2634 of file TrueType.cs.

7.149.2.9 Get1000EmKerning() [2/2]

```
PairKerning VectSharp.TrueTypeFile.Get1000EmKerning (
            int glyph1Index,
            int glyph2Index )
```

Gets the kerning between two glyphs.

Parameters

glyph1Index	The index of the first glyph of the kerning pair.
glyph2Index	The index of the second glyph of the kerning pair.

Returns

A PairKerning object containing information about how the position of each glyphs should be altered.

Definition at line 2648 of file TrueType.cs.

7.149.2.10 Get1000EmUnderlineIntersections()

```
double[] VectSharp.TrueTypeFile.Get1000EmUnderlineIntersections (
            char glyph,
            double position,
            double thickness )
```

Computes the intersections between an underline at the specified position and thickness and a glyph, in thousandths of em units.

Parameters

glyph	The glyph whose intersections with the underline will be computed.
position	The distance of the top of the underline from the baseline, in thousandths of em unit.
thickness	The thickness of the underline, in thousandths of em unit.

Returns

If the underline does not intersect the glyph, this method returns `null`. Otherwise, it returns an array containing two elements, representing the horizontal coordinates of the leftmost and rightmost intersection points.

Definition at line 2484 of file TrueType.cs.

7.149.2.11 Get1000EmUnderlinePosition()

```
double VectSharp.TrueTypeFile.Get1000EmUnderlinePosition ( )
```

Computes the distance of the top of the underline from the baseline, in thousandths of em unit.

Returns

The distance of the top of the underline from the baseline, in thousandths of em unit.

Definition at line 2422 of file TrueType.cs.

7.149.2.12 Get1000EmUnderlineThickness()

```
double VectSharp.TrueTypeFile.Get1000EmUnderlineThickness ( )
```

Computes the thickness of the underline, in thousandths of em unit.

Returns

The thickness of the underline, in thousandths of em unit.

Definition at line 2438 of file TrueType.cs.

7.149.2.13 Get1000EmWinAscent()

```
double VectSharp.TrueTypeFile.Get1000EmWinAscent ( )
```

Computes the font's Win ascent, in thousandths of em unit.

Returns

The font's Win ascent in thousandths of em unit.

Definition at line 2245 of file TrueType.cs.

7.149.2.14 Get1000EmXMax()

```
double VectSharp.TrueTypeFile.Get1000EmXMax ( )
```

Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em unit.

Returns

The maximum distance to the right of the glyph origin of the font in thousandths of em unit.

Definition at line 2302 of file TrueType.cs.

7.149.2.15 Get1000EmXMin()

```
double VectSharp.TrueTypeFile.Get1000EmXMin ( )
```

Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em unit.

Returns

The maximum distance to the left of the glyph origin of the font in thousandths of em unit.

Definition at line 2311 of file TrueType.cs.

7.149.2.16 Get1000EmYMax()

```
double VectSharp.TrueTypeFile.Get1000EmYMax ( )
```

Computes the maximum height over the baseline of the font, in thousandths of em unit.

Returns

The maximum height over the baseline of the font in thousandths of em unit.

Definition at line 2284 of file TrueType.cs.

7.149.2.17 Get1000EmYMin()

```
double VectSharp.TrueTypeFile.Get1000EmYMin ( )
```

Computes the maximum depth below the baseline of the font, in thousandths of em unit.

Returns

The maximum depth below the baseline of the font in thousandths of em unit.

Definition at line 2293 of file TrueType.cs.

7.149.2.18 GetFirstCharIndex()

```
ushort VectSharp.TrueTypeFile.GetFirstCharIndex ( )
```

Returns the index of the first character glyph represented by the font.

Returns

The index of the first character glyph represented by the font.

Definition at line 2054 of file TrueType.cs.

7.149.2.19 GetFontFamilyName()

```
string VectSharp.TrueTypeFile.GetFontFamilyName ( )
```

Obtains the font family name from the TrueType file.

Returns

The font family name, if available; `null` otherwise.

Definition at line 1980 of file TrueType.cs.

7.149.2.20 GetFontName()

```
string VectSharp.TrueTypeFile.GetFontName ( )
```

Obtains the PostScript font name from the TrueType file.

Returns

The PostScript font name, if available; `null` otherwise.

Definition at line 2035 of file TrueType.cs.

7.149.2.21 GetFullFontFamilyName()

```
string VectSharp.TrueTypeFile.GetFullFontFamilyName ( )
```

Obtains the full font family name from the TrueType file.

Returns

The full font family name, if available; `null` otherwise.

Definition at line 2007 of file TrueType.cs.

7.149.2.22 GetGlyphIndex()

```
int VectSharp.TrueTypeFile.GetGlyphIndex (
            char glyph )
```

Determines the index of the glyph corresponding to a certain character.

Parameters

glyph	The character whose glyph is sought.

Returns

The index of the glyph in the TrueType file.

Definition at line 2144 of file TrueType.cs.

7.149.2.23 GetGlyphPath() [1/2]

```
TrueTypePoint[][] VectSharp.TrueTypeFile.GetGlyphPath (
            char glyph,
            double size )
```

Get the path that describes the shape of a glyph.

Parameters

glyph	The glyph whose path is sought.
size	The font size to be used for the font coordinates.

Returns

An array of contours, each of which is itself an array of TrueType points.

Definition at line 2206 of file TrueType.cs.

7.149.2.24 GetGlyphPath() [2/2]

```
TrueTypePoint[][] VectSharp.TrueTypeFile.GetGlyphPath (
            int glyphIndex,
            double size )
```

Get the path that describes the shape of a glyph.

Parameters

glyphIndex	The index of the glyph whose path is sought.
size	The font size to be used for the font coordinates.

Returns

An array of contours, each of which is itself an array of TrueType points.

Definition at line 2195 of file TrueType.cs.

7.149.2.25 GetItalicAngle()

```
double VectSharp.TrueTypeFile.GetItalicAngle ( )
```

Computes the italic angle for the current font, in thousandths of em unit. This is computed from the vertical and is negative for text that leans forwards.

Returns

Definition at line 2464 of file TrueType.cs.

7.149.2.26 GetLastCharIndex()

```
ushort VectSharp.TrueTypeFile.GetLastCharIndex ( )
```

Returns the index of the last character glyph represented by the font.

Returns

The index of the last character glyph represented by the font.

Definition at line 2065 of file TrueType.cs.

7.149.2.27 GetNames() [1/2]

```
static bool VectSharp.TrueTypeFile.GetNames (
            Stream fs,
            out List< TrueTypeName > names )  [static]
```

Determines whether the stream contains a valid TrueType file and retrieves the list of names defined in it.

Parameters

fs	The stream containing the TrueType file.
names	The list of names that will be returned. These will only include names with TrueType identifiers 1, 4, 6, or 16.

Returns

If the stream does not contain a valid TrueType file, `false`. If the TrueType file appears valid but does not contain a name table, `true`, but *names* will be `null`. Otherwise, `true` and *names* will not be null (but it might still be empty).

Definition at line 620 of file TrueType.cs.

7.149.2.28 GetNames() [2/2]

```
static bool VectSharp.TrueTypeFile.GetNames (
            string file,
            out List< TrueTypeName > names )  [static]
```

Determines whether the file is a valid TrueType file and retrieves the list of names defined in it.

Parameters

file	The TrueType file.
names	The list of names that will be returned. These will only include names with TrueType identifiers 1, 4, 6, or 16.

Returns

If an error occurred while opening the file, or the file does not contain a valid TrueType file, `false`. If the TrueType file appears valid but does not contain a name table, `true`, but *names* will be `null`. Otherwise, `true` and *names* will not be null (but it might still be empty).

Definition at line 596 of file TrueType.cs.

7.149.2.29 GetUnitsPerEm()

```
int VectSharp.TrueTypeFile.GetUnitsPerEm ( )
```

Returns the number of units for each em in the font file. Multiplying any glyph measurement by this size should generally lead to an integer value.

Returns

The number of units for each em in the font file.

Definition at line 2455 of file TrueType.cs.

7.149.2.30 IsBold()

```
bool VectSharp.TrueTypeFile.IsBold ( )
```

Determines whether the typeface is Bold or not.

Returns

A bool indicating whether the typeface is Bold or not

Definition at line 2099 of file TrueType.cs.

7.149.2.31 IsFixedPitch()

```
bool VectSharp.TrueTypeFile.IsFixedPitch ( )
```

Determines whether the typeface is fixed-pitch (aka monospaces) or not.

Returns

A bool indicating whether the typeface is fixed-pitch (aka monospaces) or not.

Definition at line 2110 of file TrueType.cs.

7.149.2.32 IsItalic()

```
bool VectSharp.TrueTypeFile.IsItalic ( )
```

Determines whether the typeface is Italic or Oblique or not.

Returns

A bool indicating whether the typeface is Italic or Oblique or not.

Definition at line 2077 of file TrueType.cs.

7.149.2.33 IsOblique()

```
bool VectSharp.TrueTypeFile.IsOblique ( )
```

Determines whether the typeface is Oblique or not.

Returns

A bool indicating whether the typeface is Oblique or not.

Definition at line 2088 of file TrueType.cs.

7.149.2.34 IsScript()

```
bool VectSharp.TrueTypeFile.IsScript ( )
```

Determines whether the typeface is a script typeface or not.

Returns

A bool indicating whether the typeface is a script typeface or not.

Definition at line 2132 of file TrueType.cs.

7.149.2.35 IsSerif()

```
bool VectSharp.TrueTypeFile.IsSerif ( )
```

Determines whether the typeface is serifed or not.

Returns

A bool indicating whether the typeface is serifed or not.

Definition at line 2121 of file TrueType.cs.

7.149.2.36 SubsetFont()

```
TrueTypeFile VectSharp.TrueTypeFile.SubsetFont (
            string charactersToInclude,
            bool consolidateAt32 = false,
            Dictionary< char, char > outputEncoding = null )
```

Create a subset of the TrueType file, containing only the glyphs for the specified characters.

Parameters

charactersToInclude	A string containing the characters for which the glyphs should be included.
consolidateAt32	If true, the character map is rearranged so that the included glyphs start at the unicode U+0032 control point.
outputEncoding	If *consolidateAt32* is true, entries will be added to this dictionary mapping the original characters to the new map (that starts at U+0033).

Returns

Definition at line 696 of file TrueType.cs.

7.149.3 Property Documentation

7.149.3.1 FontStream

```
Stream VectSharp.TrueTypeFile.FontStream  [get]
```

A stream pointing to the TrueType file source (either on disk or in memory). Never dispose this stream directly; if you really need to, call Destroy instead.

Definition at line 46 of file TrueType.cs.

The documentation for this class was generated from the following file:

- VectSharp/TrueType.cs

7.150 VectSharp.TrueTypeFile.TrueTypeName Struct Reference

Represets a TrueType name.

Public Types

- enum NameIdentifier

  Describes a kind of TrueType name.

Public Attributes

- NameIdentifier NameId

  The kind of name represented by this struct.
- string Name

  The name represented by this struct.

7.150.1 Detailed Description

Represets a TrueType name.

Definition at line 543 of file TrueType.cs.

7.150.2 Member Enumeration Documentation

7.150.2.1 NameIdentifier

`enum VectSharp.TrueTypeFile.TrueTypeName.NameIdentifier`

Describes a kind of TrueType name.

Definition at line 548 of file TrueType.cs.

7.150.3 Member Data Documentation

7.150.3.1 Name

`string VectSharp.TrueTypeFile.TrueTypeName.Name`

The name represented by this struct.

Definition at line 579 of file TrueType.cs.

7.150.3.2 NameId

`NameIdentifier VectSharp.TrueTypeFile.TrueTypeName.NameId`

The kind of name represented by this struct.

Definition at line 574 of file TrueType.cs.

The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.151 VectSharp.TrueTypeFile.TrueTypePoint Struct Reference

Represents a point in a TrueType path description.

Public Attributes

- double X

  The horizontal coordinate of the point.
- double Y

  The vertical coordinate of the point.
- bool IsOnCurve

  Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.

7.151.1 Detailed Description

Represents a point in a TrueType path description.

Definition at line 1494 of file TrueType.cs.

7.151.2 Member Data Documentation

7.151.2.1 IsOnCurve

```
bool VectSharp.TrueTypeFile.TrueTypePoint.IsOnCurve
```

Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.

Definition at line 1509 of file TrueType.cs.

7.151.2.2 X

```
double VectSharp.TrueTypeFile.TrueTypePoint.X
```

The horizontal coordinate of the point.

Definition at line 1499 of file TrueType.cs.

7.151.2.3 Y

```
double VectSharp.TrueTypeFile.TrueTypePoint.Y
```

The vertical coordinate of the point.

Definition at line 1504 of file TrueType.cs.

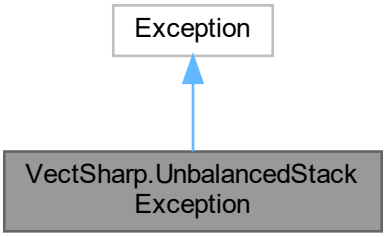
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.152 VectSharp.UnbalancedStackException Class Reference

The exception that is thrown when an unbalanced graphics state stack occurs.

Inheritance diagram for VectSharp.UnbalancedStackException:



7.152.1 Detailed Description

The exception that is thrown when an unbalanced graphics state stack occurs.

Definition at line 254 of file Graphics.cs.

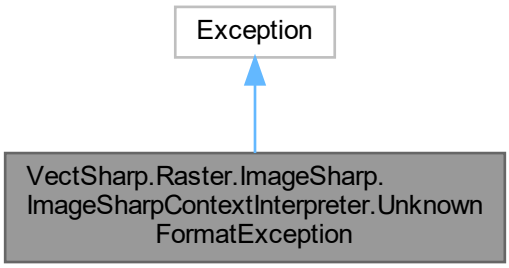
The documentation for this class was generated from the following file:

- VectSharp/Graphics.cs

7.153 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.↩ UnknownFormatException Class Reference

The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Inheritance diagram for VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException:



Properties

- string Format [get]

  The extension of the file that does not correspond to any known file format.

7.153.1 Detailed Description

The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Definition at line 1113 of file ImageSharpContext.cs.

7.153.2 Property Documentation

7.153.2.1 Format

```
string VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException.Format
[get]
```

The extension of the file that does not correspond to any known file format.

Definition at line 1118 of file ImageSharpContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Raster.ImageSharp/ImageSharpContext.cs

7.154 VectSharp.TrueTypeFile.VerticalMetrics Struct Reference

Represents the maximum heigth above and depth below the baseline of a glyph.

Public Attributes

- int YMin

  The maximum depth below the baseline of the glyph.
- int YMax

  The maximum height above the baseline of the glyph.

7.154.1 Detailed Description

Represents the maximum heigth above and depth below the baseline of a glyph.

Definition at line 2374 of file TrueType.cs.

7.154.2 Member Data Documentation

7.154.2.1 YMax

```
int VectSharp.TrueTypeFile.VerticalMetrics.YMax
```

The maximum height above the baseline of the glyph.

Definition at line 2384 of file TrueType.cs.

7.154.2.2 YMin

```
int VectSharp.TrueTypeFile.VerticalMetrics.YMin
```

The maximum depth below the baseline of the glyph.

Definition at line 2379 of file TrueType.cs.

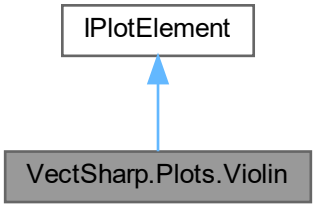
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.155 VectSharp.Plots.Violin Class Reference

A plot element that draws a violin plot.

Inheritance diagram for VectSharp.Plots.Violin:



Public Types

- enum ViolinSide

  The sides on which a violin can be drawn.

Public Member Functions

- Violin (IReadOnlyList< double > position, IReadOnlyList< double > direction, IReadOnlyList< double > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Violin instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Position `[get, set]`

  The position of the origin of the violin (e.g., the 0 in data space coordinates).
- IReadOnlyList< double > Direction `[get, set]`

  The direction along which the violin is drawn, in data space coordinates.
- double Width = 10 `[get, set]`

  The width of the violin in data space coordinates.
- bool Smooth = true `[get, set]`

  Determines whether the violin is smoothed or not.
- ViolinSide Sides = ViolinSide.Both `[get, set]`

  Determines on which side(s) the violin is drawn.
- IReadOnlyList< double > Data `[get, set]`

  The values whose distribution is displayed by the violin plot.
- int MaxBins = int.MaxValue `[get, set]`

  Maximum number of bins.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Fill = Colours.White } `[get, set]`

  Presentation attributes for the violin plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- string Tag `[get, set]`

  A tag to identify the violin in the plot.

7.155.1 Detailed Description

A plot element that draws a violin plot.

Definition at line 27 of file Violin.cs.

7.155.2 Member Enumeration Documentation

7.155.2.1 ViolinSide

`enum VectSharp.Plots.Violin.ViolinSide`

The sides on which a violin can be drawn.

Definition at line 32 of file Violin.cs.

7.155.3 Constructor & Destructor Documentation

7.155.3.1 Violin()

```
VectSharp.Plots.Violin.Violin (
            IReadOnlyList< double > position,
            IReadOnlyList< double > direction,
            IReadOnlyList< double > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Violin instance.

Parameters

position	The position of the origin of the violin (e.g., the 0 in data space coordinates).
direction	The direction along which the violin is drawn, in data space coordinates.
data	The values whose distribution is displayed by the violin plot.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 108 of file Violin.cs.

7.155.4 Member Function Documentation

7.155.4.1 Plot()

```
void VectSharp.Plots.Violin.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 117 of file Violin.cs.

7.155.5 Property Documentation

7.155.5.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.Violin.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 93 of file Violin.cs.

7.155.5.2 Data

```
IReadOnlyList<double> VectSharp.Plots.Violin.Data  [get], [set]
```

The values whose distribution is displayed by the violin plot.

Definition at line 78 of file Violin.cs.

7.155.5.3 Direction

```
IReadOnlyList<double> VectSharp.Plots.Violin.Direction  [get], [set]
```

The direction along which the violin is drawn, in data space coordinates.

Definition at line 58 of file Violin.cs.

7.155.5.4 MaxBins

```
int VectSharp.Plots.Violin.MaxBins = int.MaxValue  [get], [set]
```

Maximum number of bins.

Definition at line 83 of file Violin.cs.

7.155.5.5 Position

```
IReadOnlyList<double> VectSharp.Plots.Violin.Position  [get], [set]
```

The position of the origin of the violin (e.g., the 0 in data space coordinates).

Definition at line 53 of file Violin.cs.

7.155.5.6 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Violin.PresentationAttributes = new PlotElementPresentationA
{ Fill = Colours.White }  [get], [set]
```

Presentation attributes for the violin plot.

Definition at line 88 of file Violin.cs.

7.155.5.7 Sides

```
ViolinSide VectSharp.Plots.Violin.Sides = ViolinSide.Both  [get], [set]
```

Determines on which side(s) the violin is drawn.

Definition at line 73 of file Violin.cs.

7.155.5.8 Smooth

```
bool VectSharp.Plots.Violin.Smooth = true  [get], [set]
```

Determines whether the violin is smoothed or not.

Definition at line 68 of file Violin.cs.

7.155.5.9 Tag

```
string VectSharp.Plots.Violin.Tag  [get], [set]
```

A tag to identify the violin in the plot.

Definition at line 99 of file Violin.cs.

7.155.5.10 Width

```
double VectSharp.Plots.Violin.Width = 10  [get], [set]
```

The width of the violin in data space coordinates.

Definition at line 63 of file Violin.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Violin.cs

Chapter 8

File Documentation

8.1 AnimatedCanvas.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2022-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using System;
00021 using System.Collections.Generic;
00022
00023 namespace VectSharp.Canvas
00024 {
00025 /// <summary>
00026 /// An <see cref="Avalonia.Controls.Canvas"/> containing an animation.
00027 /// </summary>
00028     public class AnimatedCanvas :  Avalonia.Controls.Control
00029     {
00030 /// <summary>
00031 /// Defines the <see cref="CurrentFrame"/> property.
00032 /// </summary>
00033         public static readonly StyledProperty<int> CurrentFrameProperty =
     AvaloniaProperty.Register<AnimatedCanvas, int>(nameof(CurrentFrame), coerce:  (ownerObject, val) =>
00034         {
00035             AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
00036
00037             if (owner.maxFrameIndex > 0)
00038             {
00039                 if (val < owner.maxFrameIndex)
00040                 {
00041                     return val;
00042                 }
00043                 else
00044                 {
00045                     owner.IsPlaying = false;
00046                     return owner.maxFrameIndex - 1;
00047                 }
00048             }
00049             else
00050             {
00051                 return val % owner.RenderActions.Length;
00052             }
00053         });
00054
00055 /// <summary>
00056 /// The current frame in the animation.
00057 /// </summary>
```

```
00058         public int CurrentFrame
00059         {
00060             get { return GetValue(CurrentFrameProperty); }
00061             set { SetValue(CurrentFrameProperty, value); }
00062         }
00063
00064 /// <summary>
00065 /// Defines the <see cref="FrameCount"/> property.
00066 /// </summary>
00067         public static readonly DirectProperty<AnimatedCanvas, int> FrameCountProperty =
      AvaloniaProperty.RegisterDirect<AnimatedCanvas, int>(nameof(FrameCount), o => o.maxFrameIndex);
00068
00069         private int frameCount;
00070
00071 /// <summary>
00072 /// The number of frames in the animation.
00073 /// </summary>
00074         public int FrameCount
00075         {
00076             get
00077             {
00078                 return frameCount;
00079             }
00080             private set
00081             {
00082                 SetAndRaise(FrameCountProperty, ref frameCount, value);
00083             }
00084         }
00085
00086         private int maxFrameIndex { get; }
00087
00088 /// <summary>
00089 /// Defines the <see cref="FrameRate"/> property.
00090 /// </summary>
00091         public static readonly DirectProperty<AnimatedCanvas, double> FrameRateProperty =
      AvaloniaProperty.RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o => o.FrameRate);
00092
00093         private double frameRate;
00094
00095 /// <summary>
00096 /// The target frame rate of the animation.
00097 /// </summary>
00098         public double FrameRate
00099         {
00100             get
00101             {
00102                 return frameRate;
00103             }
00104             private set
00105             {
00106                 SetAndRaise(FrameRateProperty, ref frameRate, value);
00107             }
00108         }
00109
00110
00111
00112 /// <summary>
00113 /// Defines the <see cref="IsPlaying"/> property.
00114 /// </summary>
00115         public static readonly StyledProperty<bool> IsPlayingProperty =
      AvaloniaProperty.Register<AnimatedCanvas, bool>(nameof(IsPlaying), false, coerce:  (ownerObject, val)
      =>
00116         {
00117             AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
00118
00119             if (owner.IsPlaying && !val)
00120             {
00121                 owner.RefreshTimer.Dispose();
00122             }
00123             else if (!owner.IsPlaying && val)
00124             {
00125                 owner.RefreshTimer = new System.Threading.Timer(_ =>
00126                 {
00127                     _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00128                     {
00129                         owner.CurrentFrame++;
00130                     });
00131                 }, null, 0, (long)(1000.0 / owner.FrameRate));
00132             }
00133
00134             return val;
00135         });
00136
00137 /// <summary>
00138 /// The current frame in the animation.
00139 /// </summary>
00140         public bool IsPlaying
```

```
00141         {
00142             get { return GetValue(IsPlayingProperty); }
00143             set { SetValue(IsPlayingProperty, value); }
00144         }
00145 
00146 
00147         private List<RenderAction>[] RenderActions;
00148 
00149         static Avalonia.Point Origin = new Avalonia.Point(0, 0);
00150 
00151         private SolidColorBrush BackgroundBrush;
00152 
00153         private Dictionary<string, (IImage, bool)> Images;
00154 
00155         private System.Threading.Timer RefreshTimer { get; set; }
00156 
00157         static AnimatedCanvas()
00158         {
00159             AffectsRender<AnimatedCanvas>(CurrentFrameProperty);
00160         }
00161 
00162         internal AnimatedCanvas(VectSharp.Animation animation, double durationScaling, double
      frameRate, AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
00163         {
00164             Colour backgroundColour = animation.Background;
00165 
00166             this.BackgroundBrush = new SolidColorBrush(Color.FromArgb((byte)(backgroundColour.A *
      255), (byte)(backgroundColour.R * 255), (byte)(backgroundColour.G * 255), (byte)(backgroundColour.B *
      255)));
00167 
00168             this.Width = animation.Width;
00169             this.Height = animation.Height;
00170             this.Images = new Dictionary<string, (IImage, bool)>();
00171 
00172             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
00173 
00174             this.RenderActions = new List<RenderAction>[frames];
00175 
00176             for (int i = 0; i < frames; i++)
00177             {
00178                 double frameTime = i / frameRate / durationScaling * 1000;
00179 
00180                 Page pag = animation.GetFrameAtAbsolute(frameTime);
00181 
00182                 AvaloniaDrawingContext ctx = new AvaloniaDrawingContext(this.Width, this.Height,
      false, textOption, this.Images, filterOption);
00183 
00184                 pag.Graphics.CopyToIGraphicsContext(ctx);
00185                 this.RenderActions[i] = ctx.RenderActions;
00186             }
00187 
00188             if (animation.RepeatCount > 0)
00189             {
00190                 this.maxFrameIndex = animation.RepeatCount * frames;
00191             }
00192             else
00193             {
00194                 this.maxFrameIndex = 0;
00195             }
00196 
00197             this.FrameCount = frames;
00198 
00199             this.FrameRate = frameRate;
00200         }
00201 
00202 /// <inheritdoc/>
00203         public override void Render(DrawingContext context)
00204         {
00205             context.FillRectangle(this.BackgroundBrush, new Avalonia.Rect(0, 0, Width, Height));
00206 
00207             foreach (RenderAction act in this.RenderActions[this.CurrentFrame %
      this.RenderActions.Length])
00208             {
00209                 if (act.ActionType == RenderAction.ActionTypes.Path)
00210                 {
00211                     DrawingContext.PushedState?  state = null;
00212 
00213                     if (act.ClippingPath != null)
00214                     {
00215                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00216                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00217                         state = context.PushGeometryClip(act.ClippingPath);
00218                     }
00219 
00220                     using (context.PushTransform(act.Transform))
00221                     {
```

```
00222                         context.DrawGeometry(act.Fill, act.Stroke, act.Geometry);
00223                     }
00224 
00225                     if (state != null)
00226                     {
00227                         state?.Dispose();
00228                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00229                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00230                     }
00231                 }
00232                 else if (act.ActionType == RenderAction.ActionTypes.Text)
00233                 {
00234                     DrawingContext.PushedState?  state = null;
00235 
00236                     if (act.ClippingPath != null)
00237                     {
00238                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00239                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00240                         state = context.PushGeometryClip(act.ClippingPath);
00241                     }
00242 
00243                     using (context.PushTransform(act.Transform))
00244                     {
00245                         context.DrawText(act.Text, Origin);
00246                     }
00247 
00248                     if (state != null)
00249                     {
00250                         state?.Dispose();
00251                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00252                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00253                     }
00254                 }
00255                 else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
00256                 {
00257                     DrawingContext.PushedState?  state = null;
00258 
00259                     if (act.ClippingPath != null)
00260                     {
00261                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00262                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00263                         state = context.PushGeometryClip(act.ClippingPath);
00264                     }
00265 
00266                     (IImage, bool) image = Images[act.ImageId];
00267 
00268                     using (context.PushTransform(act.Transform))
00269                     {
00270                         RenderOptions.SetBitmapInterpolationMode(this, image.Item2 ?
      Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality :
      Avalonia.Media.Imaging.BitmapInterpolationMode.None);
00271                         context.DrawImage(image.Item1, act.ImageSource.Value,
      act.ImageDestination.Value);
00272                     }
00273 
00274                     if (state != null)
00275                     {
00276                         state?.Dispose();
00277                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00278                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00279                     }
00280                 }
00281             }
00282 
00283         }
00284 
00285 /// <inheritdoc/>
00286         protected override void OnDetachedFromVisualTree(VisualTreeAttachmentEventArgs e)
00287         {
00288             base.OnDetachedFromVisualTree(e);
00289             IsPlaying = false;
00290         }
00291 
00292 /// <inheritdoc/>
00293         protected override void OnAttachedToVisualTree(VisualTreeAttachmentEventArgs e)
00294         {
00295             base.OnAttachedToVisualTree(e);
00296             IsPlaying = true;
00297         }
00298     }
00299 
00300 }
```

8.2 AvaloniaContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia.Controls;
00019 using Avalonia.Controls.Shapes;
00020 using Avalonia.Media;
00021 using Avalonia.Media.Imaging;
00022 using Avalonia.Media.TextFormatting;
00023 using System;
00024 using System.Collections.Generic;
00025 using System.Linq;
00026 using System.Runtime.InteropServices;
00027 using VectSharp.Filters;
00028
00029 namespace VectSharp.Canvas
00030 {
00031     // Adapted from
      https://github.com/AvaloniaUI/Avalonia/blob/412438d334f970367834d52b92b9ed3e89f3d18c/src/Avalonia.Controls/TextBlock.cs
00032     internal readonly struct SimpleTextSource :  ITextSource
00033     {
00034         private readonly string _text;
00035         private readonly TextRunProperties _defaultProperties;
00036
00037         public SimpleTextSource(string text, TextRunProperties defaultProperties)
00038         {
00039             _text = text;
00040             _defaultProperties = defaultProperties;
00041         }
00042
00043         public TextRun GetTextRun(int textSourceIndex)
00044         {
00045             if (textSourceIndex > _text.Length)
00046             {
00047                 return new TextEndOfParagraph();
00048             }
00049
00050             var runText = _text.AsMemory(textSourceIndex);
00051
00052             if (runText.IsEmpty)
00053             {
00054                 return new TextEndOfParagraph();
00055             }
00056
00057             return new TextCharacters(runText, _defaultProperties);
00058         }
00059     }
00060
00061     internal static class MatrixUtils
00062     {
00063         public static Avalonia.Matrix ToAvaloniaMatrix(this double[,] matrix)
00064         {
00065             return new Avalonia.Matrix(matrix[0, 0], matrix[1, 0], matrix[0, 1], matrix[1, 1],
      matrix[0, 2], matrix[1, 2]);
00066         }
00067
00068         public static double[] Multiply(double[,] matrix, double[] vector)
00069         {
00070             double[] tbr = new double[2];
00071
00072             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00073             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00074
00075             return tbr;
00076         }
00077
00078         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00079         {
00080             double[,] tbr = new double[3, 3];
00081
00082             for (int i = 0; i < 3; i++)
00083             {
```

```
00084                 for (int j = 0; j < 3; j++)
00085                 {
00086                     for (int k = 0; k < 3; k++)
00087                     {
00088                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00089                     }
00090                 }
00091             }
00092 
00093             return tbr;
00094         }
00095 
00096         public static double[,] Rotate(double[,] matrix, double angle)
00097         {
00098             double[,] rotationMatrix = new double[3, 3];
00099             rotationMatrix[0, 0] = Math.Cos(angle);
00100             rotationMatrix[0, 1] = -Math.Sin(angle);
00101             rotationMatrix[1, 0] = Math.Sin(angle);
00102             rotationMatrix[1, 1] = Math.Cos(angle);
00103             rotationMatrix[2, 2] = 1;
00104 
00105             return Multiply(matrix, rotationMatrix);
00106         }
00107 
00108         public static double[,] Translate(double[,] matrix, double x, double y)
00109         {
00110             double[,] translationMatrix = new double[3, 3];
00111             translationMatrix[0, 0] = 1;
00112             translationMatrix[0, 2] = x;
00113             translationMatrix[1, 1] = 1;
00114             translationMatrix[1, 2] = y;
00115             translationMatrix[2, 2] = 1;
00116 
00117             return Multiply(matrix, translationMatrix);
00118         }
00119 
00120         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00121         {
00122             double[,] scaleMatrix = new double[3, 3];
00123             scaleMatrix[0, 0] = scaleX;
00124             scaleMatrix[1, 1] = scaleY;
00125             scaleMatrix[2, 2] = 1;
00126 
00127             return Multiply(matrix, scaleMatrix);
00128         }
00129 
00130         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00131         public static double[,] Invert(double[,] m)
00132         {
00133             double[,] tbr = new double[3, 3];
00134 
00135             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00136             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00137             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00138             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00139             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00140             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00141             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00142             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00143             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00144 
00145             return tbr;
00146         }
00147     }
00148 
00149     internal static class Utils
00150     {
00151         public static void CoerceNaNAndInfinityToZero(ref double val)
00152         {
```

```
00153             if (double.IsNaN(val) || double.IsInfinity(val) || val == double.MinValue || val ==
      double.MaxValue)
00154             {
00155                 val = 0;
00156             }
00157         }
00158
00159         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2)
00160         {
00161             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00162             {
00163                 val1 = 0;
00164             }
00165
00166             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00167             {
00168                 val2 = 0;
00169             }
00170         }
00171
00172         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4)
00173         {
00174             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00175             {
00176                 val1 = 0;
00177             }
00178
00179             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00180             {
00181                 val2 = 0;
00182             }
00183
00184             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00185             {
00186                 val3 = 0;
00187             }
00188
00189             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00190             {
00191                 val4 = 0;
00192             }
00193         }
00194
00195         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4, ref double val5, ref double val6)
00196         {
00197             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00198             {
00199                 val1 = 0;
00200             }
00201
00202             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00203             {
00204                 val2 = 0;
00205             }
00206
00207             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00208             {
00209                 val3 = 0;
00210             }
00211
00212             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00213             {
00214                 val4 = 0;
00215             }
00216
00217             if (double.IsNaN(val5) || double.IsInfinity(val5) || val5 == double.MinValue || val5 ==
      double.MaxValue)
00218             {
00219                 val5 = 0;
00220             }
00221
00222             if (double.IsNaN(val6) || double.IsInfinity(val6) || val6 == double.MinValue || val6 ==
      double.MaxValue)
00223             {
00224                 val6 = 0;
```

```
00225             }
00226         }
00227     }
00228
00229     internal class AvaloniaContext :  IGraphicsContext
00230     {
00231         public Dictionary<string, Delegate> TaggedActions { get; set; } = new Dictionary<string,
      Delegate>();
00232
00233         private bool removeTaggedActions = true;
00234
00235         public string Tag { get; set; }
00236
00237         private AvaloniaContextInterpreter.TextOptions _textOption;
00238
00239         private Avalonia.Controls.Canvas currControlElement;
00240
00241         private Stack<Avalonia.Controls.Canvas> controlElements;
00242         private FilterOption _filterOption;
00243
00244         public AvaloniaContext(double width, double height, bool removeTaggedActionsAfterExecution,
      AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
00245         {
00246             currentPath = new PathGeometry();
00247             currentFigure = new PathFigure() { IsClosed = false };
00248             figureInitialised = false;
00249             ControlItem = new Avalonia.Controls.Canvas() { Width = width, Height = height,
      ClipToBounds = true };
00250             removeTaggedActions = removeTaggedActionsAfterExecution;
00251
00252             _transform = new double[3, 3];
00253
00254             _transform[0, 0] = 1;
00255             _transform[1, 1] = 1;
00256             _transform[2, 2] = 1;
00257
00258             states = new Stack<double[,]>();
00259
00260             _textOption = textOption;
00261
00262             currControlElement = ControlItem;
00263             controlElements = new Stack<Avalonia.Controls.Canvas>();
00264             controlElements.Push(ControlItem);
00265             _filterOption = filterOption;
00266         }
00267
00268         public Avalonia.Controls.Canvas ControlItem { get; }
00269
00270         public double Width { get { return ControlItem.Width; } }
00271         public double Height { get { return ControlItem.Height; } }
00272
00273         public void Translate(double x, double y)
00274         {
00275             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00276
00277             _transform = MatrixUtils.Translate(_transform, x, y);
00278
00279             currentPath = new PathGeometry();
00280             currentFigure = new PathFigure() { IsClosed = false };
00281             figureInitialised = false;
00282         }
00283
00284         public TextBaselines TextBaseline { get; set; }
00285
00286         private void PathText(string text, double x, double y)
00287         {
00288             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00289
00290             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00291
00292             for (int j = 0; j < textPath.Segments.Count; j++)
00293             {
00294                 switch (textPath.Segments[j].Type)
00295                 {
00296                     case VectSharp.SegmentType.Move:
00297                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00298                         break;
00299                     case VectSharp.SegmentType.Line:
00300                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00301                         break;
00302                     case VectSharp.SegmentType.CubicBezier:
00303                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00304                         break;
00305                     case VectSharp.SegmentType.Close:
00306                         this.Close();
```

```
00307                       break;
00308                   }
00309               }
00310           }
00311
00312           public void StrokeText(string text, double x, double y)
00313           {
00314               Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00315
00316               PathText(text, x, y);
00317               Stroke();
00318           }
00319
00320           public void FillText(string text, double x, double y)
00321           {
00322               Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00323
00324               if (_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && Font.FontFamily.IsStandardFamily &&
      Font.FontFamily.FileName != "ZapfDingbats" && Font.FontFamily.FileName != "Symbol"))
00325               {
00326
00327                   TextBlock blk = new TextBlock() { ClipToBounds = false, Text = text, FontFamily =
      Avalonia.Media.FontFamily.Parse(FontFamily), FontSize = Font.FontSize, FontStyle =
      (Font.FontFamily.IsOblique ?  FontStyle.Oblique :  Font.FontFamily.IsItalic ?  FontStyle.Italic :
      FontStyle.Normal), FontWeight = (Font.FontFamily.IsBold ?  FontWeight.Bold :  FontWeight.Regular),
      TextAlignment = TextAlignment.Left };
00328
00329                   double top = y;
00330                   double left = x;
00331
00332                   double[,] currTransform = null;
00333                   double[,] deltaTransform = MatrixUtils.Identity;
00334
00335                   if (Font.FontFamily.TrueTypeFile != null)
00336                   {
00337                       currTransform = MatrixUtils.Translate(_transform, x, y);
00338                   }
00339
00340                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00341
00342                   if (text.StartsWith(" "))
00343                   {
00344                       var spaceMetrics = Font.MeasureTextAdvanced(" ");
00345
00346                       for (int i = 0; i < text.Length; i++)
00347                       {
00348                           if (text[i] == ' ')
00349                           {
00350                               left -= spaceMetrics.AdvanceWidth;
00351                           }
00352                           else
00353                           {
00354                               break;
00355                           }
00356                       }
00357                   }
00358
00359                   if (TextBaseline == TextBaselines.Top)
00360                   {
00361                       blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00362
00363                       if (Font.FontFamily.TrueTypeFile != null)
00364                       {
00365                           if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00366                           {
00367                               currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.WinAscent);
00368                               deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.WinAscent);
00369                           }
00370                           else
00371                           {
00372                               currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.Ascent);
00373                               deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.Ascent);
00374                           }
00375                       }
00376                   }
00377                   else if (TextBaseline == TextBaselines.Middle)
00378                   {
00379                       blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00380
00381                       if (Font.FontFamily.TrueTypeFile != null)
00382                       {
00383                           if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
```

```
00384                         {
00385                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
00386                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
00387                         }
00388                         else
00389                         {
00390                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
00391                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
00392                         }
00393 
00394                     }
00395                 }
00396                 else if (TextBaseline == TextBaselines.Baseline)
00397                 {
00398                     blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00399 
00400                     if (Font.FontFamily.TrueTypeFile != null)
00401                     {
00402                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00403                         {
00404                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent);
00405                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.YMax);
00406                         }
00407                         else
00408                         {
00409                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent);
00410                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.YMax);
00411                         }
00412                     }
00413                 }
00414                 else if (TextBaseline == TextBaselines.Bottom)
00415                 {
00416                     blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Bottom;
00417 
00418                     if (Font.FontFamily.TrueTypeFile != null)
00419                     {
00420                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00421                         {
00422                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent + metrics.Bottom);
00423                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent + metrics.Bottom);
00424                         }
00425                         else
00426                         {
00427                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent + metrics.Bottom);
00428                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent + metrics.Bottom);
00429                         }
00430                     }
00431                 }
00432 
00433                 blk.RenderTransform = new MatrixTransform(currTransform.ToAvaloniaMatrix());
00434                 blk.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00435 
00436                 Avalonia.Media.Brush foreground = null;
00437 
00438                 if (this.FillStyle is SolidColourBrush solid)
00439                 {
00440                     foreground = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00441                 }
00442                 else if (this.FillStyle is LinearGradientBrush linearGradient)
00443                 {
00444                     foreground = linearGradient.ToLinearGradientBrush(deltaTransform);
00445                 }
00446                 else if (this.FillStyle is RadialGradientBrush radialGradient)
00447                 {
00448                     foreground = radialGradient.ToRadialGradientBrush(metrics.Width +
      metrics.LeftSideBearing + metrics.RightSideBearing, deltaTransform);
00449                 }
00450 
00451                 blk.Foreground = foreground;
00452 
00453                 currControlElement.Children.Add(blk);
00454 
00455                 if (!string.IsNullOrEmpty(Tag))
```

```
00456                 {
00457                     if (TaggedActions.ContainsKey(Tag))
00458                     {
00459                         TaggedActions[Tag].DynamicInvoke(blk);
00460 
00461                         if (removeTaggedActions)
00462                         {
00463                             TaggedActions.Remove(Tag);
00464                         }
00465                     }
00466                 }
00467             }
00468             else
00469             {
00470                 PathText(text, x, y);
00471                 Fill(FillRule.NonZeroWinding);
00472             }
00473         }
00474 
00475         public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
00476         private byte StrokeAlpha = 255;
00477 
00478         public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
00479         private byte FillAlpha = 255;
00480 
00481         public void SetFillStyle((int r, int g, int b, double a) style)
00482         {
00483             FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
00484             FillAlpha = (byte)(style.a * 255);
00485         }
00486 
00487         public void SetFillStyle(Brush style)
00488         {
00489             FillStyle = style;
00490 
00491             if (style is SolidColourBrush solid)
00492             {
00493                 FillAlpha = (byte)(solid.A * 255);
00494             }
00495             else
00496             {
00497                 FillAlpha = 255;
00498             }
00499         }
00500 
00501         public void SetStrokeStyle((int r, int g, int b, double a) style)
00502         {
00503             StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
00504             StrokeAlpha = (byte)(style.a * 255);
00505         }
00506 
00507         public void SetStrokeStyle(Brush style)
00508         {
00509             StrokeStyle = style;
00510 
00511             if (style is SolidColourBrush solid)
00512             {
00513                 StrokeAlpha = (byte)(solid.A * 255);
00514             }
00515             else
00516             {
00517                 StrokeAlpha = 255;
00518             }
00519         }
00520 
00521         private double[] LineDash;
00522 
00523         public void SetLineDash(LineDash dash)
00524         {
00525             LineDash = new double[] { dash.UnitsOn, dash.UnitsOff, dash.Phase };
00526         }
00527 
00528         public void Rotate(double angle)
00529         {
00530             Utils.CoerceNaNAndInfinityToZero(ref angle);
00531 
00532             _transform = MatrixUtils.Rotate(_transform, angle);
00533 
00534             currentPath = new PathGeometry();
00535             currentFigure = new PathFigure() { IsClosed = false };
00536             figureInitialised = false;
00537         }
00538 
00539         public void Transform(double a, double b, double c, double d, double e, double f)
00540         {
00541             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
00542 
```

```
00543             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00544             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
00545 
00546             currentPath = new PathGeometry();
00547             currentFigure = new PathFigure() { IsClosed = false };
00548             figureInitialised = false;
00549         }
00550 
00551         public void Scale(double x, double y)
00552         {
00553             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00554 
00555             _transform = MatrixUtils.Scale(_transform, x, y);
00556 
00557             currentPath = new PathGeometry();
00558             currentFigure = new PathFigure() { IsClosed = false };
00559             figureInitialised = false;
00560         }
00561 
00562         private double[,] _transform;
00563 
00564         private readonly Stack<double[,]> states;
00565 
00566         public void Save()
00567         {
00568             states.Push((double[,])_transform.Clone());
00569             controlElements.Push(currControlElement);
00570         }
00571 
00572         public void Restore()
00573         {
00574             _transform = states.Pop();
00575             currControlElement = controlElements.Pop();
00576         }
00577 
00578         public double LineWidth { get; set; }
00579         public LineCaps LineCap { get; set; }
00580         public LineJoins LineJoin { get; set; }
00581 
00582         private string FontFamily;
00583         private Font _Font;
00584 
00585         public Font Font
00586         {
00587             get
00588             {
00589                 return _Font;
00590             }
00591 
00592             set
00593             {
00594                 _Font = value;
00595 
00596                 if (Font.FontFamily is ResourceFontFamily fam)
00597                 {
00598                     FontFamily = fam.ResourceName;
00599                 }
00600                 else
00601                 {
00602                     if (!Font.FontFamily.IsStandardFamily)
00603                     {
00604                         if (Font.FontFamily.TrueTypeFile != null)
00605                         {
00606                             FontFamily = Font.FontFamily.TrueTypeFile.GetFontFamilyName();
00607                         }
00608                         else
00609                         {
00610                             FontFamily = Font.FontFamily.FileName;
00611                         }
00612                     }
00613                     else
00614                     {
00615                         FontFamily = "resm:VectSharp.StandardFonts.?assembly=VectSharp#" +
      Font.FontFamily.TrueTypeFile.GetFontFamilyName();
00616                     }
00617                 }
00618             }
00619         }
00620 
00621         /*public (double Width, double Height) MeasureText(string text)
00622 {
00623 
00624 
00625 
00626 Avalonia.Media.FormattedText txt = new Avalonia.Media.FormattedText() { Text = text, Typeface = new
      Typeface(FontFamily), FontSize = Font.FontSize };
00627 
```

```
00628
00629
00630 return (txt.Bounds.Width, txt.Bounds.Height);
00631 }*/
00632
00633         private PathGeometry currentPath;
00634         private PathFigure currentFigure;
00635
00636         private bool figureInitialised = false;
00637
00638         public void MoveTo(double x, double y)
00639         {
00640             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00641
00642             if (figureInitialised)
00643             {
00644                 currentPath.Figures.Add(currentFigure);
00645             }
00646
00647             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed = false
      };
00648             figureInitialised = true;
00649         }
00650
00651         public void LineTo(double x, double y)
00652         {
00653             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00654
00655             if (figureInitialised)
00656             {
00657                 currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x, y) });
00658             }
00659             else
00660             {
00661                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed =
      false };
00662                 figureInitialised = true;
00663             }
00664         }
00665
00666         public void Rectangle(double x0, double y0, double width, double height)
00667         {
00668             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
00669
00670             if (currentFigure != null && figureInitialised)
00671             {
00672                 currentPath.Figures.Add(currentFigure);
00673             }
00674
00675             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x0, y0), IsClosed =
      false };
00676             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0) });
00677             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0 + height) });
00678             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0, y0 + height) });
00679             currentFigure.IsClosed = true;
00680
00681             currentPath.Figures.Add(currentFigure);
00682             figureInitialised = false;
00683         }
00684
00685         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00686         {
00687             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
00688
00689             if (figureInitialised)
00690             {
00691                 currentFigure.Segments.Add(new Avalonia.Media.BezierSegment() { Point1 = new
      Avalonia.Point(p1X, p1Y), Point2 = new Avalonia.Point(p2X, p2Y), Point3 = new Avalonia.Point(p3X, p3Y)
      });
00692             }
00693             else
00694             {
00695                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(p1X, p1Y), IsClosed
      = false };
00696                 figureInitialised = true;
00697             }
00698         }
00699
00700         public void Close()
00701         {
00702             currentFigure.IsClosed = true;
00703             currentPath.Figures.Add(currentFigure);
```

```
00704             figureInitialised = false;
00705         }
00706
00707         public void Stroke()
00708         {
00709             if (figureInitialised)
00710             {
00711                 currentFigure.IsClosed = false;
00712                 currentPath.Figures.Add(currentFigure);
00713             }
00714
00715             Avalonia.Media.Brush stroke = null;
00716
00717             if (this.StrokeStyle is SolidColourBrush solid)
00718             {
00719                 stroke = new SolidColorBrush(Color.FromArgb(StrokeAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00720             }
00721             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
00722             {
00723                 stroke = linearGradient.ToLinearGradientBrush();
00724             }
00725             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
00726             {
00727                 stroke = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
00728             }
00729
00730
00731             Path pth = new Path() { Fill = null, Stroke = stroke, StrokeThickness = LineWidth,
      StrokeDashArray = new Avalonia.Collections.AvaloniaList<double> { (LineDash[0] + (LineCap ==
      LineCaps.Butt ?  0 :  LineWidth)) / LineWidth, (LineDash[1] - (LineCap == LineCaps.Butt ?  0 :
      LineWidth)) / LineWidth }, StrokeDashOffset = LineDash[2] / LineWidth };
00732
00733             switch (LineCap)
00734             {
00735                 case LineCaps.Butt:
00736                     pth.StrokeLineCap = PenLineCap.Flat;
00737                     break;
00738                 case LineCaps.Round:
00739                     pth.StrokeLineCap = PenLineCap.Round;
00740                     break;
00741                 case LineCaps.Square:
00742                     pth.StrokeLineCap = PenLineCap.Square;
00743                     break;
00744             }
00745
00746             switch (LineJoin)
00747             {
00748                 case LineJoins.Bevel:
00749                     pth.StrokeJoin = PenLineJoin.Bevel;
00750                     break;
00751                 case LineJoins.Round:
00752                     pth.StrokeJoin = PenLineJoin.Round;
00753                     break;
00754                 case LineJoins.Miter:
00755                     pth.StrokeJoin = PenLineJoin.Miter;
00756                     break;
00757             }
00758
00759             pth.Data = currentPath;
00760
00761             pth.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00762             pth.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00763
00764             currControlElement.Children.Add(pth);
00765
00766             currentPath = new PathGeometry();
00767             currentFigure = new PathFigure() { IsClosed = false };
00768             figureInitialised = false;
00769
00770             if (!string.IsNullOrEmpty(Tag))
00771             {
00772                 if (TaggedActions.ContainsKey(Tag))
00773                 {
00774                     TaggedActions[Tag].DynamicInvoke(pth);
00775
00776                     if (removeTaggedActions)
00777                     {
00778                         TaggedActions.Remove(Tag);
00779                     }
00780                 }
00781             }
00782         }
00783
00784         public void Fill(FillRule fillRule)
00785         {
```

```
00786             if (figureInitialised)
00787             {
00788                 currentPath.Figures.Add(currentFigure);
00789             }
00790 
00791             Avalonia.Media.Brush fill = null;
00792 
00793             if (this.FillStyle is SolidColourBrush solid)
00794             {
00795                 fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00796             }
00797             else if (this.FillStyle is LinearGradientBrush linearGradient)
00798             {
00799                 fill = linearGradient.ToLinearGradientBrush();
00800             }
00801             else if (this.FillStyle is RadialGradientBrush radialGradient)
00802             {
00803                 fill = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
00804             }
00805 
00806             Path pth = new Path() { Fill = fill, Stroke = null };
00807 
00808             switch (fillRule)
00809             {
00810                 case FillRule.NonZeroWinding:
00811                     currentPath.FillRule = Avalonia.Media.FillRule.NonZero;
00812                     break;
00813 
00814                 case FillRule.EvenOdd:
00815                     currentPath.FillRule = Avalonia.Media.FillRule.EvenOdd;
00816                     break;
00817             }
00818 
00819             pth.Data = currentPath;
00820 
00821             pth.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00822             pth.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00823 
00824             currControlElement.Children.Add(pth);
00825 
00826             currentPath = new PathGeometry();
00827             currentFigure = new PathFigure() { IsClosed = false };
00828             figureInitialised = false;
00829 
00830             if (!string.IsNullOrEmpty(Tag))
00831             {
00832                 if (TaggedActions.ContainsKey(Tag))
00833                 {
00834                     TaggedActions[Tag].DynamicInvoke(pth);
00835 
00836                     if (removeTaggedActions)
00837                     {
00838                         TaggedActions.Remove(Tag);
00839                     }
00840                 }
00841             }
00842         }
00843 
00844         public void SetClippingPath()
00845         {
00846             if (figureInitialised)
00847             {
00848                 currentPath.Figures.Add(currentFigure);
00849             }
00850 
00851             Avalonia.Controls.Canvas newControlElement = new Avalonia.Controls.Canvas();
00852 
00853             newControlElement.Clip = currentPath;
00854 
00855             newControlElement.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00856             newControlElement.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00857 
00858             _transform = new double[3, 3];
00859 
00860             _transform[0, 0] = 1;
00861             _transform[1, 1] = 1;
00862             _transform[2, 2] = 1;
00863 
00864             currControlElement.Children.Add(newControlElement);
00865             currControlElement = newControlElement;
00866 
00867             currentPath = new PathGeometry();
00868             currentFigure = new PathFigure() { IsClosed = false };
00869             figureInitialised = false;
```

```
00870         }
00871
00872
00873         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00874         {
00875             Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
00876
00877             Image img = new Image() { Source = new CroppedBitmap(new Bitmap(image.PNGStream), new
      Avalonia.PixelRect(sourceX, sourceY, sourceWidth, sourceHeight)), Width = destinationWidth, Height =
      destinationHeight };
00878
00879             if (image.Interpolate)
00880             {
00881                 RenderOptions.SetBitmapInterpolationMode(img,
      Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality);
00882             }
00883             else
00884             {
00885                 RenderOptions.SetBitmapInterpolationMode(img,
      Avalonia.Media.Imaging.BitmapInterpolationMode.None);
00886             }
00887
00888             double[,] transf = MatrixUtils.Translate(_transform, destinationX, destinationY);
00889             img.RenderTransform = new MatrixTransform(transf.ToAvaloniaMatrix());
00890             img.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00891
00892             currControlElement.Children.Add(img);
00893
00894             if (!string.IsNullOrEmpty(Tag))
00895             {
00896                 if (TaggedActions.ContainsKey(Tag))
00897                 {
00898                     TaggedActions[Tag].DynamicInvoke(img);
00899
00900                     if (removeTaggedActions)
00901                     {
00902                         TaggedActions.Remove(Tag);
00903                     }
00904                 }
00905             }
00906         }
00907
00908         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00909         {
00910             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
00911             {
00912                 double scale = this._filterOption.RasterisationResolution;
00913
00914                 Rectangle bounds = graphics.GetBounds();
00915
00916                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00917
00918                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00919                 {
00920                     if (!this._filterOption.RasterisationResolutionRelative)
00921                     {
00922                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
00923                     }
00924
00925                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
00926                     RasterImage filtered = null;
00927
00928                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00929                     {
00930                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
00931                     }
00932
00933                     if (filter is ILocationInvariantFilter locInvFilter)
00934                     {
00935                         filtered = locInvFilter.Filter(rasterised, scale);
00936                     }
00937                     else if (filter is IFilterWithLocation filterWithLoc)
00938                     {
00939                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00940                     }
00941
00942                     if (filtered != null)
00943                     {
00944                         rasterised.Dispose();
```

```
00945
00946                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00947                     }
00948                 }
00949             }
00950             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
00951             {
00952                 double scale = this._filterOption.RasterisationResolution;
00953
00954                 Rectangle bounds = graphics.GetBounds();
00955
00956                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00957
00958                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00959                 {
00960                     if (!this._filterOption.RasterisationResolutionRelative)
00961                     {
00962                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
00963                     }
00964
00965                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
00966                     {
00967                         RasterImage filtered = null;
00968
00969                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00970                         {
00971
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
00972                         }
00973
00974                         if (filter is ILocationInvariantFilter locInvFilter)
00975                         {
00976                             filtered = locInvFilter.Filter(rasterised, scale);
00977                         }
00978                         else if (filter is IFilterWithLocation filterWithLoc)
00979                         {
00980                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00981                         }
00982
00983                         if (filtered != null)
00984                         {
00985                             rasterised.Dispose();
00986
00987                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00988                         }
00989                     }
00990                     else
00991                     {
00992                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
00993 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
00994 • Provide your own implementation of Graphics.RasterisationMethod.
00995 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
00996                     }
00997                 }
00998             }
00999             else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
01000             {
01001                 graphics.CopyToIGraphicsContext(this);
01002             }
01003             else
01004             {
01005
01006             }
01007         }
01008     }
01009
01010     internal class RenderCanvas :  Control
01011     {
01012         private List<RenderAction> RenderActions;
01013         private List<RenderAction> TaggedRenderActions;
01014
01015         private SolidColorBrush BackgroundBrush;
01016
01017         static Avalonia.Point Origin = new Avalonia.Point(0, 0);
01018
01019         public Dictionary<string, (IImage, bool)> Images;
01020
01021         public void BringToFront(RenderAction action)
01022         {
01023             this.RenderActions.Remove(action);
```

```
01024             this.RenderActions.Add(action);
01025
01026             if (!string.IsNullOrEmpty(action.Tag))
01027             {
01028                 int index = this.TaggedRenderActions.IndexOf(action);
01029                 if (index >= 0)
01030                 {
01031                     this.TaggedRenderActions.RemoveAt(index);
01032                     this.TaggedRenderActions.Insert(0, action);
01033                 }
01034             }
01035         }
01036
01037         public void SendToBack(RenderAction action)
01038         {
01039             this.RenderActions.Remove(action);
01040             this.RenderActions.Insert(0, action);
01041
01042             if (!string.IsNullOrEmpty(action.Tag))
01043             {
01044                 int index = this.TaggedRenderActions.IndexOf(action);
01045                 if (index >= 0)
01046                 {
01047                     this.TaggedRenderActions.RemoveAt(index);
01048                     this.TaggedRenderActions.Add(action);
01049                 }
01050             }
01051         }
01052
01053         public RenderCanvas(Graphics content, Colour backgroundColour, double width, double height,
      Dictionary<string, Delegate> taggedActions, bool removeTaggedActionsAfterExecution,
      AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
01054         {
01055             this.BackgroundBrush = new SolidColorBrush(Color.FromArgb((byte)(backgroundColour.A *
      255), (byte)(backgroundColour.R * 255), (byte)(backgroundColour.G * 255), (byte)(backgroundColour.B *
      255)));
01056             this.Width = width;
01057             this.Height = height;
01058             this.Images = new Dictionary<string, (IImage, bool)>();
01059             AvaloniaDrawingContext ctx = new AvaloniaDrawingContext(this.Width, this.Height,
      removeTaggedActionsAfterExecution, textOption, this.Images, filterOption);
01060             foreach (KeyValuePair<string, Delegate> action in taggedActions)
01061             {
01062                 ctx.TaggedActions.Add(action.Key, (Func<RenderAction,
      IEnumerable<RenderAction>)action.Value);
01063             }
01064
01065             content.CopyToIGraphicsContext(ctx);
01066             this.RenderActions = ctx.RenderActions;
01067
01068             this.TaggedRenderActions = new List<RenderAction>();
01069
01070             for (int i = this.RenderActions.Count - 1; i >= 0; i--)
01071             {
01072                 RenderActions[i].InternalParent = this;
01073                 if (!string.IsNullOrEmpty(this.RenderActions[i].Tag))
01074                 {
01075                     TaggedRenderActions.Add(this.RenderActions[i]);
01076                 }
01077             }
01078
01079             this.PointerPressed += PointerPressedAction;
01080             this.PointerReleased += PointerReleasedAction;
01081             this.PointerMoved += PointerMoveAction;
01082             this.PointerExited += PointerLeaveAction;
01083         }
01084
01085
01086         private int CurrentPressedAction = -1;
01087         private void PointerPressedAction(object sender, Avalonia.Input.PointerPressedEventArgs e)
01088         {
01089             Avalonia.Point position = e.GetPosition(this);
01090
01091             for (int i = 0; i < TaggedRenderActions.Count; i++)
01092             {
01093                 if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01094                 {
01095                     Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01096
01097                     if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01098                     {
01099                         if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01100                         {
```

```
01101                               TaggedRenderActions[i].FirePointerPressed(e);
01102                               CurrentPressedAction = i;
01103                               break;
01104                           }
01105                       }
01106                       else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01107                       {
01108                           if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01109                           {
01110                               TaggedRenderActions[i].FirePointerPressed(e);
01111                               CurrentPressedAction = i;
01112                               break;
01113                           }
01114                       }
01115                       else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01116                       {
01117                           if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01118                           {
01119                               TaggedRenderActions[i].FirePointerPressed(e);
01120                               CurrentPressedAction = i;
01121                               break;
01122                           }
01123                       }
01124                   }
01125               }
01126           }
01127
01128           private void PointerReleasedAction(object sender, Avalonia.Input.PointerReleasedEventArgs e)
01129           {
01130               if (CurrentPressedAction >= 0)
01131               {
01132                   TaggedRenderActions[CurrentPressedAction].FirePointerReleased(e);
01133                   CurrentPressedAction = -1;
01134               }
01135               else
01136               {
01137                   Avalonia.Point position = e.GetPosition(this);
01138
01139                   for (int i = 0; i < TaggedRenderActions.Count; i++)
01140                   {
01141                       if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01142                       {
01143                           Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01144
01145                           if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01146                           {
01147                               if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01148                               {
01149                                   TaggedRenderActions[i].FirePointerReleased(e);
01150                                   break;
01151                               }
01152                           }
01153                           else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01154                           {
01155                               if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01156                               {
01157                                   TaggedRenderActions[i].FirePointerReleased(e);
01158                                   break;
01159                               }
01160                           }
01161                           else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01162                           {
01163                               if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01164                               {
01165                                   TaggedRenderActions[i].FirePointerReleased(e);
01166                                   break;
01167                               }
01168                           }
01169                       }
01170                   }
01171               }
01172           }
01173
01174           private int CurrentOverAction = -1;
01175           private void PointerMoveAction(object sender, Avalonia.Input.PointerEventArgs e)
01176           {
01177               Avalonia.Point position = e.GetPosition(this);
01178
01179               bool found = false;
```

```
01180
01181             for (int i = 0; i < TaggedRenderActions.Count; i++)
01182             {
01183                 if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01184                 {
01185                     Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01186
01187                     if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01188                     {
01189                         if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01190                         {
01191                             found = true;
01192
01193                             if (CurrentOverAction != i)
01194                             {
01195                                 if (CurrentOverAction >= 0)
01196                                 {
01197                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01198                                 }
01199                                 CurrentOverAction = i;
01200                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01201                             }
01202
01203                             break;
01204                         }
01205                     }
01206                     else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01207                     {
01208                         if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01209                         {
01210                             found = true;
01211
01212                             if (CurrentOverAction != i)
01213                             {
01214                                 if (CurrentOverAction >= 0)
01215                                 {
01216                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01217                                 }
01218                                 CurrentOverAction = i;
01219                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01220                             }
01221
01222                             break;
01223                         }
01224                     }
01225                     else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01226                     {
01227                         if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01228                         {
01229                             found = true;
01230
01231                             if (CurrentOverAction != i)
01232                             {
01233                                 if (CurrentOverAction >= 0)
01234                                 {
01235                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01236                                 }
01237                                 CurrentOverAction = i;
01238                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01239                             }
01240
01241                             break;
01242                         }
01243                     }
01244                 }
01245             }
01246
01247             if (!found)
01248             {
01249                 if (CurrentOverAction >= 0)
01250                 {
01251                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01252                 }
01253                 CurrentOverAction = -1;
01254             }
01255         }
01256
01257         private void PointerLeaveAction(object sender, Avalonia.Input.PointerEventArgs e)
01258         {
01259             if (CurrentOverAction >= 0)
01260             {
```

```
01261                 TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01262             }
01263             CurrentOverAction = -1;
01264         }
01265 
01266 
01267         public override void Render(DrawingContext context)
01268         {
01269             context.FillRectangle(this.BackgroundBrush, new Avalonia.Rect(0, 0, Width, Height));
01270 
01271             foreach (RenderAction act in this.RenderActions)
01272             {
01273                 if (act.ActionType == RenderAction.ActionTypes.Path)
01274                 {
01275                     DrawingContext.PushedState?  state = null;
01276 
01277                     if (act.ClippingPath != null)
01278                     {
01279                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01280                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01281                         state = context.PushGeometryClip(act.ClippingPath);
01282                     }
01283 
01284                     using (context.PushTransform(act.Transform))
01285                     {
01286                         context.DrawGeometry(act.Fill, act.Stroke, act.Geometry);
01287                     }
01288 
01289                     if (state != null)
01290                     {
01291                         state?.Dispose();
01292                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01293                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01294                     }
01295                 }
01296                 else if (act.ActionType == RenderAction.ActionTypes.Text)
01297                 {
01298                     DrawingContext.PushedState?  state = null;
01299 
01300                     if (act.ClippingPath != null)
01301                     {
01302                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01303                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01304                         state = context.PushGeometryClip(act.ClippingPath);
01305                     }
01306 
01307                     using (context.PushTransform(act.Transform))
01308                     {
01309                         context.DrawText(act.Text, Origin);
01310                     }
01311 
01312                     if (state != null)
01313                     {
01314                         state?.Dispose();
01315                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01316                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01317                     }
01318                 }
01319                 else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
01320                 {
01321                     DrawingContext.PushedState?  state = null;
01322 
01323                     if (act.ClippingPath != null)
01324                     {
01325                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01326                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01327                         state = context.PushGeometryClip(act.ClippingPath);
01328                     }
01329 
01330                     (IImage, bool) image = Images[act.ImageId];
01331 
01332                     using (context.PushTransform(act.Transform))
01333                     {
01334                         using (context.PushRenderOptions(new RenderOptions() { BitmapInterpolationMode
      = image.Item2 ?  Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality :
      Avalonia.Media.Imaging.BitmapInterpolationMode.None }))
01335                         {
01336                             context.DrawImage(image.Item1, act.ImageSource.Value,
      act.ImageDestination.Value);
01337                         }
01338                     }
01339 
```

```
01340                     if (state != null)
01341                     {
01342                         state?.Dispose();
01343                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01344                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01345                     }
01346                 }
01347             }
01348         }
01349     }
01350
01351
01352 /// <summary>
01353 /// Represents a light-weight rendering action.
01354 /// </summary>
01355     public class RenderAction
01356     {
01357 /// <summary>
01358 /// Types of rendering actions.
01359 /// </summary>
01360         public enum ActionTypes
01361         {
01362 /// <summary>
01363 /// The render action represents a path object.
01364 /// </summary>
01365             Path,
01366
01367 /// <summary>
01368 /// The render action represents a text object.
01369 /// </summary>
01370             Text,
01371
01372 /// <summary>
01373 /// The render action represents a raster image.
01374 /// </summary>
01375             RasterImage
01376         }
01377
01378 /// <summary>
01379 /// Type of the rendering action.
01380 /// </summary>
01381         public ActionTypes ActionType { get; private set; }
01382
01383 /// <summary>
01384 /// Geometry that needs to be rendered (null if the action type is <see cref="ActionTypes.Text"/>).
      If you change this, you need to invalidate the <see cref="Parent"/>'s visual.
01385 /// </summary>
01386         public Geometry Geometry { get; set; }
01387
01388 /// <summary>
01389 /// Text that needs to be rendered (null if the action type is <see cref="ActionTypes.Path"/>).  If
      you change this, you need to invalidate the <see cref="Parent"/>'s visual and
01390 /// you should also change the <see cref="Layout"/> property.
01391 /// </summary>
01392         public Avalonia.Media.FormattedText Text { get; set; }
01393
01394 /// <summary>
01395 /// Text layout, used for hit testing (null if the action type is <see cref="ActionTypes.Path"/>).  If
      you change this, you need to invalidate the <see cref="Parent"/>'s visual and
01396 /// you should also change the <see cref="Text"/> property.
01397 /// </summary>
01398         public Avalonia.Media.TextFormatting.TextLayout Layout { get; set; }
01399
01400 /// <summary>
01401 /// Rendering stroke (null if the action type is <see cref="ActionTypes.Text"/> or if the rendered
      action only has a <see cref="Fill"/>).  If you change this, you need to invalidate the <see
      cref="Parent"/>'s visual.
01402 /// </summary>
01403         public Pen Stroke { get; set; }
01404
01405
01406         private IBrush fill;
01407
01408 /// <summary>
01409 /// Rendering fill (null if the rendered action only has a <see cref="Stroke"/>).  If you change this,
      you need to invalidate the <see cref="Parent"/>'s visual.
01410 /// </summary>
01411         public IBrush Fill
01412         {
01413             get
01414             {
01415                 return fill;
01416             }
01417             set
01418             {
01419                 fill = value;
```

```
01420
01421                 if (Text != null)
01422                 {
01423                     Text.SetForegroundBrush(fill);
01424                 }
01425             }
01426         }
01427
01428 /// <summary>
01429 /// Univocal identifier of the image that needs to be drawn.
01430 /// </summary>
01431         public string ImageId { get; set; }
01432
01433 /// <summary>
01434 /// The source rectangle of the image.
01435 /// </summary>
01436         public Avalonia.Rect?  ImageSource { get; set; }
01437
01438 /// <summary>
01439 /// The destination rectangle of the image.
01440 /// </summary>
01441         public Avalonia.Rect?  ImageDestination { get; set; }
01442
01443 /// <summary>
01444 /// The current clipping path.
01445 /// </summary>
01446         public Geometry ClippingPath { get; set; }
01447
01448         private Avalonia.Matrix _transform = Avalonia.Matrix.Identity;
01449
01450 /// <summary>
01451 /// Inverse transformation matrix.
01452 /// </summary>
01453         public Avalonia.Matrix InverseTransform { get; private set; } = Avalonia.Matrix.Identity;
01454
01455 /// <summary>
01456 /// Rendering transformation matrix.  If you change this, you need to invalidate the <see
      cref="Parent"/>'s visual.
01457 /// </summary>
01458         public Avalonia.Matrix Transform
01459         {
01460             get { return _transform; }
01461             set
01462             {
01463                 _transform = value;
01464                 InverseTransform = _transform.Invert();
01465             }
01466         }
01467
01468 /// <summary>
01469 /// A tag to access the <see cref="RenderAction"/>.
01470 /// </summary>
01471         public string Tag { get; set; }
01472
01473         internal RenderCanvas InternalParent { get; set; }
01474
01475 /// <summary>
01476 /// The container of this <see cref="RenderAction"/>.
01477 /// </summary>
01478         public Control Parent
01479         {
01480             get
01481             {
01482                 return InternalParent;
01483             }
01484         }
01485
01486 /// <summary>
01487 /// Raised when the pointer enters the area covered by the <see cref="RenderAction"/>.
01488 /// </summary>
01489         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerEntered;
01490
01491 /// <summary>
01492 /// Raised when the pointer leaves the area covered by the <see cref="RenderAction"/>.
01493 /// </summary>
01494         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerExited;
01495
01496 /// <summary>
01497 /// Raised when the pointer is pressed while over the area covered by the <see cref="RenderAction"/>.
01498 /// </summary>
01499         public event EventHandler<Avalonia.Input.PointerPressedEventArgs> PointerPressed;
01500
01501 /// <summary>
01502 /// Raised when the pointer is released after a <see cref="PointerPressed"/> event.
01503 /// </summary>
01504         public event EventHandler<Avalonia.Input.PointerReleasedEventArgs> PointerReleased;
01505
```

```
01506
01507         internal void FirePointerEntered(Avalonia.Input.PointerEventArgs e)
01508         {
01509             this.PointerEntered?.Invoke(this, e);
01510         }
01511
01512         internal void FirePointerExited(Avalonia.Input.PointerEventArgs e)
01513         {
01514             this.PointerExited?.Invoke(this, e);
01515         }
01516
01517         internal void FirePointerPressed(Avalonia.Input.PointerPressedEventArgs e)
01518         {
01519             this.PointerPressed?.Invoke(this, e);
01520         }
01521
01522         internal void FirePointerReleased(Avalonia.Input.PointerReleasedEventArgs e)
01523         {
01524             this.PointerReleased?.Invoke(this, e);
01525         }
01526
01527         private RenderAction()
01528         {
01529
01530         }
01531
01532 /// <summary>
01533 /// Creates a new <see cref="RenderAction"/> representing a path.
01534 /// </summary>
01535 /// <param name="geometry">The geometry to be rendered.</param>
01536 /// <param name="stroke">The stroke of the path (can be null).</param>
01537 /// <param name="fill">The fill of the path (can be null).</param>
01538 /// <param name="transform">The transform that will be applied to the path.</param>
01539 /// <param name="clippingPath">The clipping path.</param>
01540 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01541 /// <returns>A new <see cref="RenderAction"/> representing a path.</returns>
01542         public static RenderAction PathAction(Geometry geometry, Pen stroke, IBrush fill,
      Avalonia.Matrix transform, Geometry clippingPath, string tag = null)
01543         {
01544             return new RenderAction()
01545             {
01546                 ActionType = ActionTypes.Path,
01547                 Geometry = geometry,
01548                 Stroke = stroke,
01549                 Fill = fill,
01550                 Transform = transform,
01551                 ClippingPath = clippingPath,
01552                 Tag = tag
01553             };
01554         }
01555
01556 /// <summary>
01557 /// Creates a new <see cref="RenderAction"/> representing text.
01558 /// </summary>
01559 /// <param name="text">The text to be rendered.</param>
01560 /// <param name="layout">The text layout used for hit testing.</param>
01561 /// <param name="fill">The fill of the text (can be null).</param>
01562 /// <param name="transform">The transform that will be applied to the text.</param>
01563 /// <param name="clippingPath">The clipping path.</param>
01564 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01565 /// <returns>A new <see cref="RenderAction"/> representing text.</returns>
01566         public static RenderAction TextAction(Avalonia.Media.FormattedText text,
      Avalonia.Media.TextFormatting.TextLayout layout, IBrush fill, Avalonia.Matrix transform, Geometry
      clippingPath, string tag = null)
01567         {
01568             return new RenderAction()
01569             {
01570                 ActionType = ActionTypes.Text,
01571                 Text = text,
01572                 Stroke = null,
01573                 Fill = fill,
01574                 Transform = transform,
01575                 ClippingPath = clippingPath,
01576                 Tag = tag,
01577                 Layout = layout
01578             };
01579         }
01580
01581 /// <summary>
01582 /// Creates a new <see cref="RenderAction"/> representing an image.
01583 /// </summary>
01584 /// <param name="imageId">The univocal identifier of the image to draw.</param>
01585 /// <param name="sourceRect">The source rectangle of the image.</param>
01586 /// <param name="destinationRect">The destination rectangle of the image.</param>
01587 /// <param name="transform">The transform that will be applied to the image.</param>
```

```
01588 /// <param name="clippingPath">The clipping path.</param>
01589 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01590 /// <returns>A new <see cref="RenderAction"/> representing an image.</returns>
01591         public static RenderAction ImageAction(string imageId, Avalonia.Rect sourceRect, Avalonia.Rect
      destinationRect, Avalonia.Matrix transform, Geometry clippingPath, string tag = null)
01592         {
01593             return new RenderAction()
01594             {
01595                 ActionType = ActionTypes.RasterImage,
01596                 ImageId = imageId,
01597                 ImageSource = sourceRect,
01598                 ImageDestination = destinationRect,
01599                 Transform = transform,
01600                 ClippingPath = clippingPath,
01601                 Tag = tag
01602             };
01603         }
01604
01605 /// <summary>
01606 /// Brings the render action to the front of the rendering queue.  This method can only be invoked
      after the output has been fully initialised.
01607 /// </summary>
01608         public void BringToFront()
01609         {
01610             this.InternalParent.BringToFront(this);
01611         }
01612
01613 /// <summary>
01614 /// Brings the render action to the back of the rendering queue.  This method can only be invoked
      after the output has been fully initialised.
01615 /// </summary>
01616         public void SendToBack()
01617         {
01618             this.InternalParent.SendToBack(this);
01619         }
01620     }
01621
01622
01623     internal class AvaloniaDrawingContext :  IGraphicsContext
01624     {
01625         public Dictionary<string, Func<RenderAction, IEnumerable<RenderAction>> TaggedActions { get;
      set; } = new Dictionary<string, Func<RenderAction, IEnumerable<RenderAction>>();
01626
01627         private bool removeTaggedActions = true;
01628
01629         public string Tag { get; set; }
01630
01631         AvaloniaContextInterpreter.TextOptions _textOption;
01632
01633         private Dictionary<string, (IImage, bool)> Images;
01634
01635         private Geometry _clippingPath;
01636         private Stack<Geometry> clippingPaths;
01637         private FilterOption _filterOption;
01638
01639         public AvaloniaDrawingContext(double width, double height, bool
      removeTaggedActionsAfterExecution, AvaloniaContextInterpreter.TextOptions textOption,
      Dictionary<string, (IImage, bool)> images, FilterOption filterOption)
01640         {
01641             this.Images = images;
01642
01643             currentPath = new PathGeometry();
01644             currentFigure = new PathFigure() { IsClosed = false };
01645             figureInitialised = false;
01646
01647             RenderActions = new List<RenderAction>();
01648             removeTaggedActions = removeTaggedActionsAfterExecution;
01649
01650             Width = width;
01651             Height = height;
01652
01653             _transform = new double[3, 3];
01654
01655             _transform[0, 0] = 1;
01656             _transform[1, 1] = 1;
01657             _transform[2, 2] = 1;
01658
01659             states = new Stack<double[,]>();
01660
01661             _textOption = textOption;
01662
01663             _clippingPath = null;
01664             clippingPaths = new Stack<Geometry>();
01665             clippingPaths.Push(_clippingPath);
01666
01667             _filterOption = filterOption;
```

```
01668         }
01669
01670         public List<RenderAction> RenderActions { get; set; }
01671
01672         public double Width { get; private set; }
01673         public double Height { get; private set; }
01674
01675         public void Translate(double x, double y)
01676         {
01677             _transform = MatrixUtils.Translate(_transform, x, y);
01678
01679             currentPath = new PathGeometry();
01680             currentFigure = new PathFigure() { IsClosed = false };
01681             figureInitialised = false;
01682         }
01683
01684         public TextBaselines TextBaseline { get; set; }
01685
01686         private void PathText(string text, double x, double y)
01687         {
01688             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01689
01690             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
01691
01692             for (int j = 0; j < textPath.Segments.Count; j++)
01693             {
01694                 switch (textPath.Segments[j].Type)
01695                 {
01696                     case VectSharp.SegmentType.Move:
01697                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
01698                         break;
01699                     case VectSharp.SegmentType.Line:
01700                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
01701                         break;
01702                     case VectSharp.SegmentType.CubicBezier:
01703                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
01704                         break;
01705                     case VectSharp.SegmentType.Close:
01706                         this.Close();
01707                         break;
01708                 }
01709             }
01710         }
01711
01712         public void StrokeText(string text, double x, double y)
01713         {
01714             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01715
01716             PathText(text, x, y);
01717             Stroke();
01718         }
01719
01720         public void FillText(string text, double x, double y)
01721         {
01722             if (!Font.EnableKerning)
01723             {
01724                 FillSimpleText(text, x, y);
01725             }
01726             else
01727             {
01728                 List<(string, Point)> tSpans = new List<(string, Point)>();
01729
01730                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
01731                 Point currentKerning = new Point();
01732
01733                 Point currentGlyphPlacementDelta = new Point();
01734                 Point currentGlyphAdvanceDelta = new Point();
01735                 Point nextGlyphPlacementDelta = new Point();
01736                 Point nextGlyphAdvanceDelta = new Point();
01737
01738                 for (int i = 0; i < text.Length; i++)
01739                 {
01740                     if (i < text.Length - 1)
01741                     {
01742                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01743                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01744                         nextGlyphAdvanceDelta = new Point();
01745                         nextGlyphPlacementDelta = new Point();
01746
01747                         TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01748
01749                         if (kerning != null)
01750                         {
01751                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
```

```
kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01752                           currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01753
01754                           nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01755                           nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01756                       }
01757                   }
01758
01759                   if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
01760                   {
01761                       if (currentRun.Length > 0)
01762                       {
01763                           tSpans.Add((currentRun.ToString(), currentKerning));
01764
01765                           tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
01766
01767                           currentRun.Clear();
01768                           currentKerning = new Point((currentGlyphAdvanceDelta.X -
currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01769                       }
01770                       else
01771                       {
01772                           tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
currentKerning.Y)));
01773
01774                           currentRun.Clear();
01775                           currentKerning = new Point((currentGlyphAdvanceDelta.X -
currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01776                       }
01777                   }
01778                   else
01779                   {
01780                       currentRun.Append(text[i]);
01781                   }
01782               }
01783
01784               if (currentRun.Length > 0)
01785               {
01786                   tSpans.Add((currentRun.ToString(), currentKerning));
01787               }
01788
01789               double currX = x;
01790               double currY = y;
01791
01792               Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
01793
01794               if (TextBaseline == TextBaselines.Top)
01795               {
01796                   if (Font.FontFamily.TrueTypeFile != null)
01797                   {
01798                       currY += fullMetrics.Top;
01799                   }
01800               }
01801               else if (TextBaseline == TextBaselines.Middle)
01802               {
01803                   if (Font.FontFamily.TrueTypeFile != null)
01804                   {
01805                       currY += (fullMetrics.Top + fullMetrics.Bottom) * 0.5;
01806                   }
01807               }
01808               else if (TextBaseline == TextBaselines.Bottom)
01809               {
01810                   if (Font.FontFamily.TrueTypeFile != null)
01811                   {
01812                       currY += fullMetrics.Bottom;
01813                   }
01814               }
01815
01816               TextBaseline = TextBaselines.Baseline;
01817
01818               for (int i = 0; i < tSpans.Count; i++)
01819               {
01820                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
01821
01822                   if (i == 0)
01823                   {
01824                       FillSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
tSpans[i].Item2.Y);
01825                   }
```

```
01826                     else
01827                     {
01828                         FillSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
      fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
01829                     }
01830 
01831 
01832                     currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
01833                     currY += tSpans[i].Item2.Y;
01834                 }
01835             }
01836         }
01837 
01838         public void FillSimpleText(string text, double x, double y)
01839         {
01840             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01841 
01842             if (_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && Font.FontFamily.IsStandardFamily &&
      Font.FontFamily.FileName != "ZapfDingbats" && Font.FontFamily.FileName != "Symbol"))
01843             {
01844                 Typeface typeface = new Typeface(Avalonia.Media.FontFamily.Parse(FontFamily),
      (Font.FontFamily.IsOblique ?  FontStyle.Oblique :  Font.FontFamily.IsItalic ?  FontStyle.Italic :
      FontStyle.Normal), (Font.FontFamily.IsBold ?  FontWeight.Bold :  FontWeight.Regular));
01845 
01846                 Avalonia.Media.FormattedText txt = new Avalonia.Media.FormattedText(text,
      System.Globalization.CultureInfo.InvariantCulture, FlowDirection.LeftToRight, typeface, Font.FontSize,
      null);
01847 
01848                 double top = y;
01849                 double left = x;
01850 
01851                 double[,] currTransform = null;
01852                 double[,] deltaTransform = MatrixUtils.Identity;
01853 
01854                 if (Font.FontFamily.TrueTypeFile != null)
01855                 {
01856                     currTransform = MatrixUtils.Translate(_transform, x, y);
01857                 }
01858 
01859                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
01860 
01861                 if (text.StartsWith(" "))
01862                 {
01863                     var spaceMetrics = Font.MeasureTextAdvanced(" ");
01864 
01865                     for (int i = 0; i < text.Length; i++)
01866                     {
01867                         if (text[i] == ' ')
01868                         {
01869                             left -= spaceMetrics.AdvanceWidth;
01870                         }
01871                         else
01872                         {
01873                             break;
01874                         }
01875                     }
01876                 }
01877 
01878 
01879                 if (TextBaseline == TextBaselines.Top)
01880                 {
01881                     if (Font.FontFamily.TrueTypeFile != null)
01882                     {
01883                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01884                         {
01885                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.WinAscent);
01886                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.WinAscent);
01887                         }
01888                         else
01889                         {
01890                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.Ascent);
01891                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.Ascent);
01892                         }
01893 
01894                     }
01895                 }
01896                 else if (TextBaseline == TextBaselines.Middle)
01897                 {
01898                     if (Font.FontFamily.TrueTypeFile != null)
01899                     {
01900                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01901                         {
```

```
01902                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
01903                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
01904                       }
01905                       else
01906                       {
01907                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
01908                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
01909                       }
01910                   }
01911               }
01912               else if (TextBaseline == TextBaselines.Baseline)
01913               {
01914                   if (Font.FontFamily.TrueTypeFile != null)
01915                   {
01916                       if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01917                       {
01918                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.WinAscent);
01919                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.YMax);
01920                       }
01921                       else
01922                       {
01923                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.Ascent);
01924                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.YMax);
01925                       }
01926                   }
01927               }
01928               else if (TextBaseline == TextBaselines.Bottom)
01929               {
01930                   if (Font.FontFamily.TrueTypeFile != null)
01931                   {
01932                       if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01933                       {
01934                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.WinAscent + metrics.Bottom);
01935                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.WinAscent + metrics.Bottom);
01936                       }
01937                       else
01938                       {
01939                           currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.Ascent + metrics.Bottom);
01940                           deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.Ascent + metrics.Bottom);
01941                       }
01942                   }
01943               }
01944
01945               Avalonia.Media.Brush fill = null;
01946
01947               if (this.FillStyle is SolidColourBrush solid)
01948               {
01949                   fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
01950               }
01951               else if (this.FillStyle is LinearGradientBrush linearGradient)
01952               {
01953                   fill = linearGradient.ToLinearGradientBrush(deltaTransform);
01954               }
01955               else if (this.FillStyle is RadialGradientBrush radialGradient)
01956               {
01957                   fill = radialGradient.ToRadialGradientBrush(metrics.Width +
      metrics.LeftSideBearing + metrics.RightSideBearing, deltaTransform);
01958               }
01959
01960               txt.SetForegroundBrush(fill);
01961
01962               TextRunProperties runProperties = new GenericTextRunProperties(typeface,
      Font.FontSize);
01963
01964               TextLayout lay = new TextLayout(new SimpleTextSource(text, runProperties), new
      GenericTextParagraphProperties(runProperties));
01965
01966               RenderAction act = RenderAction.TextAction(txt, lay, fill,
      currTransform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
01967
01968               if (!string.IsNullOrEmpty(Tag))
01969               {
01970                   if (TaggedActions.ContainsKey(Tag))
01971                   {
```

```
01972                       IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
01973
01974                       foreach (RenderAction action in actions)
01975                       {
01976                           RenderActions.Add(action);
01977                       }
01978
01979                       if (removeTaggedActions)
01980                       {
01981                           TaggedActions.Remove(Tag);
01982                       }
01983                   }
01984                   else
01985                   {
01986                       RenderActions.Add(act);
01987                   }
01988               }
01989               else if (TaggedActions.ContainsKey(""))
01990               {
01991                   IEnumerable<RenderAction> actions = TaggedActions[""](act);
01992
01993                   foreach (RenderAction action in actions)
01994                   {
01995                       RenderActions.Add(action);
01996                   }
01997               }
01998               else
01999               {
02000                   RenderActions.Add(act);
02001               }
02002           }
02003           else
02004           {
02005               PathText(text, x, y);
02006               Fill(FillRule.NonZeroWinding);
02007           }
02008       }
02009
02010       public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
02011       private byte StrokeAlpha = 255;
02012
02013       public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
02014       private byte FillAlpha = 255;
02015
02016       public void SetFillStyle((int r, int g, int b, double a) style)
02017       {
02018           FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
02019           FillAlpha = (byte)(style.a * 255);
02020       }
02021
02022       public void SetFillStyle(Brush style)
02023       {
02024           FillStyle = style;
02025
02026           if (style is SolidColourBrush solid)
02027           {
02028               FillAlpha = (byte)(solid.A * 255);
02029           }
02030           else
02031           {
02032               FillAlpha = 255;
02033           }
02034       }
02035
02036       public void SetStrokeStyle((int r, int g, int b, double a) style)
02037       {
02038           StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
02039           StrokeAlpha = (byte)(style.a * 255);
02040       }
02041
02042       public void SetStrokeStyle(Brush style)
02043       {
02044           StrokeStyle = style;
02045
02046           if (style is SolidColourBrush solid)
02047           {
02048               StrokeAlpha = (byte)(solid.A * 255);
02049           }
02050           else
02051           {
02052               StrokeAlpha = 255;
02053           }
02054       }
02055
02056       private double[] LineDash;
02057
02058       public void SetLineDash(LineDash dash)
```

```
02059         {
02060             LineDash = new double[] { dash.UnitsOn, dash.UnitsOff, dash.Phase };
02061         }
02062 
02063         public void Rotate(double angle)
02064         {
02065             Utils.CoerceNaNAndInfinityToZero(ref angle);
02066 
02067             _transform = MatrixUtils.Rotate(_transform, angle);
02068 
02069             currentPath = new PathGeometry();
02070             currentFigure = new PathFigure() { IsClosed = false };
02071             figureInitialised = false;
02072         }
02073 
02074         public void Transform(double a, double b, double c, double d, double e, double f)
02075         {
02076             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
02077 
02078             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
02079             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
02080 
02081             currentPath = new PathGeometry();
02082             currentFigure = new PathFigure() { IsClosed = false };
02083             figureInitialised = false;
02084         }
02085 
02086         public void Scale(double x, double y)
02087         {
02088             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02089 
02090             _transform = MatrixUtils.Scale(_transform, x, y);
02091 
02092             currentPath = new PathGeometry();
02093             currentFigure = new PathFigure() { IsClosed = false };
02094             figureInitialised = false;
02095         }
02096 
02097         private double[,] _transform;
02098 
02099         private readonly Stack<double[,]> states;
02100 
02101         public void Save()
02102         {
02103             states.Push((double[,])_transform.Clone());
02104             clippingPaths.Push(_clippingPath?.Clone());
02105         }
02106 
02107         public void Restore()
02108         {
02109             _transform = states.Pop();
02110             _clippingPath = clippingPaths.Pop();
02111         }
02112 
02113         public double LineWidth { get; set; }
02114         public LineCaps LineCap { get; set; }
02115         public LineJoins LineJoin { get; set; }
02116 
02117         private string FontFamily;
02118         private Font _Font;
02119 
02120         public Font Font
02121         {
02122             get
02123             {
02124                 return _Font;
02125             }
02126 
02127             set
02128             {
02129                 _Font = value;
02130 
02131                 if (!Font.FontFamily.IsStandardFamily)
02132                 {
02133                     if (Font.FontFamily is ResourceFontFamily fam)
02134                     {
02135                         FontFamily = fam.ResourceName;
02136                     }
02137                     else
02138                     {
02139                         if (Font.FontFamily.TrueTypeFile != null)
02140                         {
02141                             FontFamily = Font.FontFamily.TrueTypeFile.GetFontFamilyName();
02142                         }
02143                         else
02144                         {
02145                             FontFamily = Font.FontFamily.FileName;
```

```
02146                         }
02147                     }
02148                 }
02149                 else
02150                 {
02151                     FontFamily = "resm:VectSharp.StandardFonts.?assembly=VectSharp#" +
      Font.FontFamily.TrueTypeFile.GetFontFamilyName();
02152                 }
02153             }
02154         }
02155 
02156         private PathGeometry currentPath;
02157         private PathFigure currentFigure;
02158 
02159         private bool figureInitialised = false;
02160 
02161         public void MoveTo(double x, double y)
02162         {
02163             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02164 
02165             if (figureInitialised)
02166             {
02167                 currentPath.Figures.Add(currentFigure);
02168             }
02169 
02170             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed = false
      };
02171             figureInitialised = true;
02172         }
02173 
02174         public void LineTo(double x, double y)
02175         {
02176             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02177 
02178             if (figureInitialised)
02179             {
02180                 currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x, y) });
02181             }
02182             else
02183             {
02184                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed =
      false };
02185                 figureInitialised = true;
02186             }
02187         }
02188 
02189         public void Rectangle(double x0, double y0, double width, double height)
02190         {
02191             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
02192 
02193             if (currentFigure != null && figureInitialised)
02194             {
02195                 currentPath.Figures.Add(currentFigure);
02196             }
02197 
02198             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x0, y0), IsClosed =
      false };
02199             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0) });
02200             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0 + height) });
02201             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0, y0 + height) });
02202             currentFigure.IsClosed = true;
02203 
02204             currentPath.Figures.Add(currentFigure);
02205             figureInitialised = false;
02206         }
02207 
02208         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
02209         {
02210             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
02211 
02212             if (figureInitialised)
02213             {
02214                 currentFigure.Segments.Add(new Avalonia.Media.BezierSegment() { Point1 = new
      Avalonia.Point(p1X, p1Y), Point2 = new Avalonia.Point(p2X, p2Y), Point3 = new Avalonia.Point(p3X, p3Y)
      });
02215             }
02216             else
02217             {
02218                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(p1X, p1Y), IsClosed
      = false };
02219                 figureInitialised = true;
02220             }
```

```
02221         }
02222
02223         public void Close()
02224         {
02225             currentFigure.IsClosed = true;
02226             currentPath.Figures.Add(currentFigure);
02227             figureInitialised = false;
02228         }
02229
02230         public void Stroke()
02231         {
02232             if (figureInitialised)
02233             {
02234                 currentFigure.IsClosed = false;
02235                 currentPath.Figures.Add(currentFigure);
02236             }
02237
02238             Avalonia.Media.Brush stroke = null;
02239
02240             if (this.StrokeStyle is SolidColourBrush solid)
02241             {
02242                 stroke = new SolidColorBrush(Color.FromArgb(StrokeAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
02243             }
02244             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
02245             {
02246                 stroke = linearGradient.ToLinearGradientBrush();
02247             }
02248             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
02249             {
02250                 stroke = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
02251             }
02252
02253             Pen pen = new Pen(stroke,
02254                     LineWidth,
02255                     new DashStyle(new double[] { (LineDash[0] + (LineCap == LineCaps.Butt ?  0 :
      LineWidth)) / LineWidth, (LineDash[1] - (LineCap == LineCaps.Butt ?  0 :  LineWidth)) / LineWidth },
      LineDash[2] / LineWidth));
02256
02257             switch (LineCap)
02258             {
02259                 case LineCaps.Butt:
02260                     pen.LineCap = PenLineCap.Flat;
02261                     break;
02262                 case LineCaps.Round:
02263                     pen.LineCap = PenLineCap.Round;
02264                     break;
02265                 case LineCaps.Square:
02266                     pen.LineCap = PenLineCap.Square;
02267                     break;
02268             }
02269
02270             switch (LineJoin)
02271             {
02272                 case LineJoins.Bevel:
02273                     pen.LineJoin = PenLineJoin.Bevel;
02274                     break;
02275                 case LineJoins.Round:
02276                     pen.LineJoin = PenLineJoin.Round;
02277                     break;
02278                 case LineJoins.Miter:
02279                     pen.LineJoin = PenLineJoin.Miter;
02280                     break;
02281             }
02282
02283             RenderAction act = RenderAction.PathAction(currentPath, pen, null,
      _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02284
02285             if (!string.IsNullOrEmpty(Tag))
02286             {
02287                 if (TaggedActions.ContainsKey(Tag))
02288                 {
02289                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02290
02291                     foreach (RenderAction action in actions)
02292                     {
02293                         RenderActions.Add(action);
02294                     }
02295
02296                     if (removeTaggedActions)
02297                     {
02298                         TaggedActions.Remove(Tag);
02299                     }
02300                 }
02301                 else
02302                 {
02303                     RenderActions.Add(act);
```

```
02304                 }
02305             }
02306             else if (TaggedActions.ContainsKey(""))
02307             {
02308                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02309 
02310                 foreach (RenderAction action in actions)
02311                 {
02312                     RenderActions.Add(action);
02313                 }
02314             }
02315             else
02316             {
02317                 RenderActions.Add(act);
02318             }
02319 
02320             currentPath = new PathGeometry();
02321             currentFigure = new PathFigure() { IsClosed = false };
02322             figureInitialised = false;
02323         }
02324 
02325         public void Fill(FillRule fillRule)
02326         {
02327             if (figureInitialised)
02328             {
02329                 currentPath.Figures.Add(currentFigure);
02330             }
02331 
02332             Avalonia.Media.Brush fill = null;
02333 
02334             if (this.FillStyle is SolidColourBrush solid)
02335             {
02336                 fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
    (byte)(solid.G * 255), (byte)(solid.B * 255)));
02337             }
02338             else if (this.FillStyle is LinearGradientBrush linearGradient)
02339             {
02340                 fill = linearGradient.ToLinearGradientBrush();
02341             }
02342             else if (this.FillStyle is RadialGradientBrush radialGradient)
02343             {
02344                 fill = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
02345             }
02346 
02347             switch (fillRule)
02348             {
02349                 case FillRule.NonZeroWinding:
02350                     currentPath.FillRule = Avalonia.Media.FillRule.NonZero;
02351                     break;
02352 
02353                 case FillRule.EvenOdd:
02354                     currentPath.FillRule = Avalonia.Media.FillRule.EvenOdd;
02355                     break;
02356             }
02357 
02358             RenderAction act = RenderAction.PathAction(currentPath, null, fill,
    _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02359 
02360             if (!string.IsNullOrEmpty(Tag))
02361             {
02362                 if (TaggedActions.ContainsKey(Tag))
02363                 {
02364                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02365 
02366                     foreach (RenderAction action in actions)
02367                     {
02368                         RenderActions.Add(action);
02369                     }
02370 
02371                     if (removeTaggedActions)
02372                     {
02373                         TaggedActions.Remove(Tag);
02374                     }
02375                 }
02376                 else
02377                 {
02378                     RenderActions.Add(act);
02379                 }
02380             }
02381             else if (TaggedActions.ContainsKey(""))
02382             {
02383                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02384 
02385                 foreach (RenderAction action in actions)
02386                 {
02387                     RenderActions.Add(action);
02388                 }
```

```
02389             }
02390             else
02391             {
02392                 RenderActions.Add(act);
02393             }
02394 
02395             currentPath = new PathGeometry();
02396             currentFigure = new PathFigure() { IsClosed = false };
02397             figureInitialised = false;
02398         }
02399 
02400         private static void TransformGeometry(PathGeometry geo, double[,] transf)
02401         {
02402             for (int i = 0; i < geo.Figures.Count; i++)
02403             {
02404                 double[] tP = MatrixUtils.Multiply(transf, new double[] { geo.Figures[i].StartPoint.X,
      geo.Figures[i].StartPoint.Y });
02405                 geo.Figures[i].StartPoint = new Avalonia.Point(tP[0], tP[1]);
02406 
02407                 for (int j = 0; j < geo.Figures[i].Segments.Count; j++)
02408                 {
02409                     if (geo.Figures[i].Segments[j] is LineSegment lS)
02410                     {
02411                         tP = MatrixUtils.Multiply(transf, new double[] { lS.Point.X, lS.Point.Y });
02412                         lS.Point = new Avalonia.Point(tP[0], tP[1]);
02413                     }
02414                     else if (geo.Figures[i].Segments[j] is BezierSegment bS)
02415                     {
02416                         tP = MatrixUtils.Multiply(transf, new double[] { bS.Point1.X, bS.Point1.Y });
02417                         double[] tP2 = MatrixUtils.Multiply(transf, new double[] { bS.Point2.X,
      bS.Point2.Y });
02418                         double[] tP3 = MatrixUtils.Multiply(transf, new double[] { bS.Point3.X,
      bS.Point3.Y });
02419 
02420                         bS.Point1 = new Avalonia.Point(tP[0], tP[1]);
02421                         bS.Point2 = new Avalonia.Point(tP2[0], tP2[1]);
02422                         bS.Point3 = new Avalonia.Point(tP3[0], tP3[1]);
02423                     }
02424                 }
02425             }
02426         }
02427 
02428         public void SetClippingPath()
02429         {
02430             if (figureInitialised)
02431             {
02432                 currentPath.Figures.Add(currentFigure);
02433             }
02434 
02435             TransformGeometry(currentPath, _transform);
02436 
02437             if (_clippingPath == null)
02438             {
02439                 _clippingPath = currentPath;
02440             }
02441             else
02442             {
02443                 _clippingPath = new CombinedGeometry(GeometryCombineMode.Intersect, _clippingPath,
      currentPath);
02444             }
02445 
02446             currentPath = new PathGeometry();
02447             currentFigure = new PathFigure() { IsClosed = false };
02448             figureInitialised = false;
02449         }
02450 
02451         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
02452         {
02453             Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
02454 
02455             if (!this.Images.ContainsKey(image.Id))
02456             {
02457                 Bitmap bmp = new Bitmap(image.PNGStream);
02458                 this.Images.Add(image.Id, (bmp, image.Interpolate));
02459             }
02460 
02461             RenderAction act = RenderAction.ImageAction(image.Id, new Avalonia.Rect(sourceX, sourceY,
      sourceWidth, sourceHeight), new Avalonia.Rect(destinationX, destinationY, destinationWidth,
      destinationHeight), _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02462 
02463             if (!string.IsNullOrEmpty(Tag))
02464             {
02465                 if (TaggedActions.ContainsKey(Tag))
02466                 {
```

```
02467                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02468
02469                     foreach (RenderAction action in actions)
02470                     {
02471                         RenderActions.Add(action);
02472                     }
02473
02474                     if (removeTaggedActions)
02475                     {
02476                         TaggedActions.Remove(Tag);
02477                     }
02478                 }
02479                 else
02480                 {
02481                     RenderActions.Add(act);
02482                 }
02483             }
02484             else if (TaggedActions.ContainsKey(""))
02485             {
02486                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02487
02488                 foreach (RenderAction action in actions)
02489                 {
02490                     RenderActions.Add(action);
02491                 }
02492             }
02493             else
02494             {
02495                 RenderActions.Add(act);
02496             }
02497         }
02498
02499         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
02500         {
02501             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
02502             {
02503                 double scale = this._filterOption.RasterisationResolution;
02504
02505                 Rectangle bounds = graphics.GetBounds();
02506
02507                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
02508
02509                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02510                 {
02511                     if (!this._filterOption.RasterisationResolutionRelative)
02512                     {
02513                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02514                     }
02515
02516                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
02517                     RasterImage filtered = null;
02518
02519                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
02520                     {
02521                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
02522                     }
02523
02524                     if (filter is ILocationInvariantFilter locInvFilter)
02525                     {
02526                         filtered = locInvFilter.Filter(rasterised, scale);
02527                     }
02528                     else if (filter is IFilterWithLocation filterWithLoc)
02529                     {
02530                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02531                     }
02532
02533                     if (filtered != null)
02534                     {
02535                         rasterised.Dispose();
02536
02537                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02538                     }
02539                 }
02540             }
02541             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
02542             {
02543                 double scale = this._filterOption.RasterisationResolution;
02544
02545                 Rectangle bounds = graphics.GetBounds();
02546
02547                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
```

```
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
02548
02549                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02550                 {
02551                     if (!this._filterOption.RasterisationResolutionRelative)
02552                     {
02553                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02554                     }
02555
02556                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
02557                     {
02558                         RasterImage filtered = null;
02559
02560                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
02561                         {
02562
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
02563                         }
02564
02565                         if (filter is ILocationInvariantFilter locInvFilter)
02566                         {
02567                             filtered = locInvFilter.Filter(rasterised, scale);
02568                         }
02569                         else if (filter is IFilterWithLocation filterWithLoc)
02570                         {
02571                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02572                         }
02573
02574                         if (filtered != null)
02575                         {
02576                             rasterised.Dispose();
02577
02578                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02579                         }
02580                     }
02581                     else
02582                     {
02583                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
02584 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
02585 • Provide your own implementation of Graphics.RasterisationMethod.
02586 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
02587                     }
02588                 }
02589             }
02590             else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
02591             {
02592                 graphics.CopyToIGraphicsContext(this);
02593             }
02594             else
02595             {
02596
02597             }
02598         }
02599     }
02600
02601
02602 /// <summary>
02603 /// Contains methods to render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/>.
02604 /// </summary>
02605     public static class AvaloniaContextInterpreter
02606     {
02607 /// <summary>
02608 /// Defines whether text items should be converted into paths when drawing.
02609 /// </summary>
02610         public enum TextOptions
02611         {
02612 /// <summary>
02613 /// Converts all text items into paths.
02614 /// </summary>
02615             AlwaysConvert,
02616
02617 /// <summary>
02618 /// Converts all text items into paths, with the exception of those that use a standard font.
02619 /// </summary>
02620             ConvertIfNecessary,
02621
02622 /// <summary>
02623 /// Does not convert any text items into paths.
02624 /// </summary>
02625             NeverConvert
02626         }
02627
02628 /// <summary>
```

```
02629 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/>.
02630 /// </summary>
02631 /// <param name="page">The <see cref="Page"/> to render.</param>
02632 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02633 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02634 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
02635         public static Avalonia.Controls.Canvas PaintToCanvas(this Page page, TextOptions textOption =
      TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02636         {
02637             filterOption = filterOption ??  FilterOption.Default;
02638 
02639             AvaloniaContext ctx = new AvaloniaContext(page.Width, page.Height, true, textOption,
      filterOption);
02640             page.Graphics.CopyToIGraphicsContext(ctx);
02641             ctx.ControlItem.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A *
      255), (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B *
      255)));
02642             return ctx.ControlItem;
02643         }
02644 
02645 /// <summary>
02646 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02647 /// </summary>
02648 /// <param name="page">The <see cref="Page"/> to render.</param>
02649 /// <param name="graphicsAsControls">If this is true, each graphics object (e.g.  paths, text...)  is
      rendered as a separate <see cref="Avalonia.Controls.Control"/>.  Otherwise, they are directly rendered
      onto the drawing context (which is faster, but does not allow interactivity).</param>
02650 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02651 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02652 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02653         public static Avalonia.Controls.Control PaintToCanvas(this Page page, bool graphicsAsControls,
      TextOptions textOption = TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02654         {
02655             filterOption = filterOption ??  FilterOption.Default;
02656 
02657             if (graphicsAsControls)
02658             {
02659                 Avalonia.Controls.Canvas tbr = page.PaintToCanvas(textOption, filterOption);
02660                 tbr.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A * 255),
      (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B * 255)));
02661                 return tbr;
02662             }
02663             else
02664             {
02665                 return new RenderCanvas(page.Graphics, page.Background, page.Width, page.Height, new
      Dictionary<string, Delegate>(), true, textOption, filterOption);
02666             }
02667         }
02668 
02669 /// <summary>
02670 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02671 /// </summary>
02672 /// <param name="page">The <see cref="Page"/> to render.</param>
02673 /// <param name="graphicsAsControls">If this is true, each graphics object (e.g.  paths, text...)  is
      rendered as a separate <see cref="Avalonia.Controls.Control"/>.  Otherwise, they are directly rendered
      onto the drawing context (which is faster, but does not allow interactivity).</param>
02674 /// <param name="taggedActions">A <see cref="Dictionary{String, Delegate}"/> containing the <see
      cref="Action"/>s that will be performed on items with the corresponding tag.
02675 /// If <paramref name="graphicsAsControls"/> is true, the delegates should be voids that accept one
      parameter of type <see cref="TextBlock"/> or <see cref="Path"/> (depending on the tagged item),
      otherwise, they should accept one parameter of type <see cref="RenderAction"/> and return an <see
      cref="IEnumerable{RenderAction}"/> of the actions that will actually be performed.</param>
02676 /// <param name="removeTaggedActionsAfterExecution">Whether the <see cref="Action"/>s should be
      removed from <paramref name="taggedActions"/> after their execution.  Set to false if the same <see
      cref="Action"/> should be performed on multiple items with the same tag.</param>
02677 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02678 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02679 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02680         public static Avalonia.Controls.Control PaintToCanvas(this Page page, bool graphicsAsControls,
      Dictionary<string, Delegate> taggedActions, bool removeTaggedActionsAfterExecution = true, TextOptions
      textOption = TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02681         {
02682             filterOption = filterOption ??  FilterOption.Default;
02683 
02684             if (graphicsAsControls)
02685             {
02686                 Avalonia.Controls.Canvas tbr = page.PaintToCanvas(taggedActions,
      removeTaggedActionsAfterExecution, textOption, filterOption);
```

```
02687                 tbr.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A * 255),
      (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B * 255)));
02688                 return tbr;
02689             }
02690             else
02691             {
02692                 return new RenderCanvas(page.Graphics, page.Background, page.Width, page.Height,
      taggedActions, removeTaggedActionsAfterExecution, textOption, filterOption);
02693             }
02694         }
02695 
02696 /// <summary>
02697 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02698 /// </summary>
02699 /// <param name="page">The <see cref="Page"/> to render.</param>
02700 /// <param name="taggedActions">A <see cref="Dictionary{String, Delegate}"/> containing the <see
      cref="Action"/>s that will be performed on items with the corresponding tag.
02701 /// The delegates should accept one parameter of type <see cref="TextBlock"/> or <see cref="Path"/>
      (depending on the tagged item).</param>
02702 /// <param name="removeTaggedActionsAfterExecution">Whether the <see cref="Action"/>s should be
      removed from <paramref name="taggedActions"/> after their execution.  Set to false if the same <see
      cref="Action"/> should be performed on multiple items with the same tag.</param>
02703 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02704 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02705 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02706         public static Avalonia.Controls.Canvas PaintToCanvas(this Page page, Dictionary<string,
      Delegate> taggedActions, bool removeTaggedActionsAfterExecution = true, TextOptions textOption =
      TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02707         {
02708             filterOption = filterOption ??  FilterOption.Default;
02709 
02710             AvaloniaContext ctx = new AvaloniaContext(page.Width, page.Height,
      removeTaggedActionsAfterExecution, textOption, filterOption)
02711             {
02712                 TaggedActions = taggedActions
02713             };
02714             page.Graphics.CopyToIGraphicsContext(ctx);
02715             ctx.ControlItem.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A *
      255), (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B *
      255)));
02716             return ctx.ControlItem;
02717         }
02718 
02719 /// <summary>
02720 /// Render an <see cref="Animation"/> to an <see cref="Avalonia.Controls.Control"/>.
02721 /// </summary>
02722 /// <param name="animation">The <see cref="Animation"/> to render.</param>
02723 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
      (fps).</param>
02724 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
02725 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02726 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02727 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered animation.</returns>
02728         public static AnimatedCanvas PaintToAnimatedCanvas(this Animation animation, double frameRate
      = 60, double durationScaling = 1, TextOptions textOption = TextOptions.ConvertIfNecessary,
      FilterOption filterOption = default)
02729         {
02730             filterOption = filterOption ??  FilterOption.Default;
02731 
02732             return new AnimatedCanvas(animation, durationScaling, frameRate, textOption,
      filterOption);
02733         }
02734 
02735         internal static Avalonia.Media.LinearGradientBrush ToLinearGradientBrush(this
      LinearGradientBrush brush, double[,] transformMatrix = null)
02736         {
02737             Point start = brush.StartPoint;
02738             Point end = brush.EndPoint;
02739 
02740             if (transformMatrix != null)
02741             {
02742                 double[,] inverse = MatrixUtils.Invert(transformMatrix);
02743 
02744                 double[] startVec = MatrixUtils.Multiply(inverse, new double[] { start.X, start.Y });
02745                 double[] endVec = MatrixUtils.Multiply(inverse, new double[] { end.X, end.Y });
02746 
02747                 start = new Point(startVec[0], startVec[1]);
02748                 end = new Point(endVec[0], endVec[1]);
02749             }
02750 
```

```
02751             Avalonia.Media.LinearGradientBrush tbr = new Avalonia.Media.LinearGradientBrush()
02752             {
02753                 SpreadMethod = GradientSpreadMethod.Pad,
02754                 StartPoint = new Avalonia.RelativePoint(start.X, start.Y,
    Avalonia.RelativeUnit.Absolute),
02755                 EndPoint = new Avalonia.RelativePoint(end.X, end.Y, Avalonia.RelativeUnit.Absolute)
02756             };
02757
02758             Avalonia.Media.GradientStops stops = new Avalonia.Media.GradientStops();
02759             stops.AddRange(from el in brush.GradientStops select new
    Avalonia.Media.GradientStop(Color.FromArgb((byte)(el.Colour.A * 255), (byte)(el.Colour.R * 255),
    (byte)(el.Colour.G * 255), (byte)(el.Colour.B * 255)), el.Offset));
02760
02761             tbr.GradientStops = stops;
02762
02763             return tbr;
02764         }
02765
02766         internal static Avalonia.Media.RadialGradientBrush ToRadialGradientBrush(this
    RadialGradientBrush brush, double objectWidth, double[,] transformMatrix = null)
02767         {
02768             Point focus = brush.FocalPoint;
02769             Point centre = brush.Centre;
02770
02771             if (transformMatrix != null)
02772             {
02773                 double[,] inverse = MatrixUtils.Invert(transformMatrix);
02774
02775                 double[] focusVec = MatrixUtils.Multiply(inverse, new double[] { focus.X, focus.Y });
02776                 double[] centreVec = MatrixUtils.Multiply(inverse, new double[] { centre.X, centre.Y
    });
02777
02778                 focus = new Point(focusVec[0], focusVec[1]);
02779                 centre = new Point(centreVec[0], centreVec[1]);
02780             }
02781
02782             Avalonia.Media.RadialGradientBrush tbr = new Avalonia.Media.RadialGradientBrush()
02783             {
02784                 SpreadMethod = GradientSpreadMethod.Pad,
02785                 Center = new Avalonia.RelativePoint(centre.X, centre.Y,
    Avalonia.RelativeUnit.Absolute),
02786                 GradientOrigin = new Avalonia.RelativePoint(focus.X, focus.Y,
    Avalonia.RelativeUnit.Absolute),
02787                 Radius = brush.Radius / objectWidth
02788             };
02789
02790             Avalonia.Media.GradientStops stops = new Avalonia.Media.GradientStops();
02791             stops.AddRange(from el in brush.GradientStops select new
    Avalonia.Media.GradientStop(Color.FromArgb((byte)(el.Colour.A * 255), (byte)(el.Colour.R * 255),
    (byte)(el.Colour.G * 255), (byte)(el.Colour.B * 255)), el.Offset));
02792             tbr.GradientStops = stops;
02793
02794             return tbr;
02795         }
02796     }
02797 }
```

8.3 RenderingParameters.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019
00020 namespace VectSharp.Canvas
00021 {
00022     internal class RenderingParameters :  IEquatable<RenderingParameters>
00023     {
00024         public static float Tolerance = 1e-5f;
```

```
00025         public float Left { get; }
00026         public float Top { get; }
00027         public float Width { get; }
00028         public float Height { get; }
00029         public float Scale { get; }
00030         public int RenderWidth { get; }
00031         public int RenderHeight { get; }
00032 
00033         public RenderingParameters(float left, float top, float width, float height, float scale, int
      renderWidth, int renderHeight)
00034         {
00035             this.Left = left;
00036             this.Top = top;
00037             this.Width = width;
00038             this.Height = height;
00039             this.Scale = scale;
00040             this.RenderWidth = renderWidth;
00041             this.RenderHeight = renderHeight;
00042         }
00043 
00044         public RenderingParameters Clone()
00045         {
00046             return new RenderingParameters(this.Left, this.Top, this.Width, this.Height, this.Scale,
      this.RenderWidth, this.RenderHeight);
00047         }
00048 
00049         public bool Equals(RenderingParameters other)
00050         {
00051             if (!object.ReferenceEquals(other, null))
00052             {
00053                 return this.Scale == other.Scale && this.Left == other.Left && this.Top == other.Top
      && this.Width == other.Width && this.Height == other.Height && this.RenderWidth == other.RenderWidth
      && this.RenderHeight == other.RenderHeight;
00054             }
00055             else
00056             {
00057                 return false;
00058             }
00059         }
00060 
00061         public override bool Equals(object obj)
00062         {
00063             if (obj is RenderingParameters other)
00064             {
00065                 return this.Equals(other);
00066             }
00067             else
00068             {
00069                 return false;
00070             }
00071         }
00072 
00073         public override int GetHashCode()
00074         {
00075             int hash = 13;
00076 
00077             unchecked
00078             {
00079                 hash = (hash * 7) + this.Left.GetHashCode();
00080                 hash = (hash * 7) + this.Top.GetHashCode();
00081                 hash = (hash * 7) + this.Width.GetHashCode();
00082                 hash = (hash * 7) + this.Height.GetHashCode();
00083                 hash = (hash * 7) + this.Scale.GetHashCode();
00084                 hash = (hash * 7) + this.RenderWidth.GetHashCode();
00085                 hash = (hash * 7) + this.RenderHeight.GetHashCode();
00086             }
00087 
00088             return hash;
00089         }
00090 
00091         public static bool operator ==(RenderingParameters param1, RenderingParameters param2)
00092         {
00093             if (object.ReferenceEquals(param1, null))
00094             {
00095                 return object.ReferenceEquals(param2, null);
00096             }
00097             else
00098             {
00099                 return param1.Equals(param2);
00100             }
00101         }
00102 
00103         public static bool operator !=(RenderingParameters param1, RenderingParameters param2)
00104         {
00105             if (!object.ReferenceEquals(param1, null) && !object.ReferenceEquals(param2, null))
00106             {
00107                 return param1.Scale != param2.Scale || param1.Left != param2.Left || param1.Top !=
```

```
      param2.Top || param1.Width != param2.Width || param1.Height != param2.Height || param1.RenderWidth !=
      param2.RenderWidth || param1.RenderHeight != param2.RenderHeight;
00108             }
00109             else
00110             {
00111                 return !(object.ReferenceEquals(param1, null) && object.ReferenceEquals(param2,
      null));
00112             }
00113         }
00114 
00115         public bool GoodEnough(RenderingParameters other)
00116         {
00117             if (this.Scale == other.Scale)
00118             {
00119                 return (this.Left <= other.Left && (this.Left + this.Width - other.Left - other.Width)
      / (this.Left + this.Width + other.Left + other.Width) >= -Tolerance && this.Top <= other.Top &&
      (this.Top + this.Height - other.Top - other.Height) / (this.Top + this.Height + other.Top +
      other.Height) >= -Tolerance);
00120             }
00121             else
00122             {
00123                 return false;
00124             }
00125         }
00126     }
00127 }
```

8.4 SkiaBitmap.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using Avalonia.Media.Imaging;
00021 using SkiaSharp;
00022 using System;
00023 using System.Linq.Expressions;
00024 using System.Reflection;
00025 
00026 namespace VectSharp.Canvas
00027 {
00028     internal class SkiaBitmap :  IImage, IDisposable
00029     {
00030         public SKCanvas SKCanvas { get; }
00031         public SKBitmap Bitmap { get; }
00032         public RenderTargetBitmap AvaloniaBitmap { get; set; }
00033         public Avalonia.Size Size { get; }
00034         public int Width { get; }
00035         public int Height { get; }
00036         public object Lock { get; }
00037 
00038 
00039         static Func<RenderTargetBitmap, SKBitmap> GetBitmap;
00040         static Func<RenderTargetBitmap, object> GetLock;
00041 
00042         static SkiaBitmap()
00043         {
00044             ParameterExpression bitmapParameter = Expression.Parameter(typeof(RenderTargetBitmap));
00045 
00046             PropertyInfo platformImplProperty = typeof(RenderTargetBitmap).GetProperty("PlatformImpl",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.DeclaredOnly);
00047             MemberExpression getPlatformImpl = Expression.Property(bitmapParameter,
      platformImplProperty);
00048             MemberExpression getItem = Expression.Property(getPlatformImpl, "Item");
00049 
00050             Type renderTargetImpl =
      typeof(Avalonia.Skia.SkiaPlatform).Assembly.GetType("Avalonia.Skia.WriteableBitmapImpl");
00051 
```

```
00052             MemberExpression getBitmap = Expression.Field(Expression.Convert(getItem,
      renderTargetImpl), "_bitmap");
00053             MemberExpression getLock = Expression.Field(Expression.Convert(getItem, renderTargetImpl),
      "_lock");
00054 
00055             GetBitmap = Expression.Lambda<Func<RenderTargetBitmap, SKBitmap>>(getBitmap,
      bitmapParameter).Compile();
00056             GetLock = Expression.Lambda<Func<RenderTargetBitmap, object>>(getLock,
      bitmapParameter).Compile();
00057         }
00058 
00059         public SkiaBitmap(int width, int height)
00060         {
00061             this.AvaloniaBitmap = new RenderTargetBitmap(new PixelSize(width, height), new Vector(96,
      96));
00062             this.Lock = GetLock(this.AvaloniaBitmap);
00063             this.Bitmap = GetBitmap(this.AvaloniaBitmap);
00064 
00065             this.SKCanvas = new SKCanvas(this.Bitmap);
00066             this.Size = new Avalonia.Size(width, height);
00067             this.Width = width;
00068             this.Height = height;
00069         }
00070 
00071         private bool disposedValue;
00072 
00073         public void Draw(DrawingContext context, Rect sourceRect, Rect destRect)
00074         {
00075             context.DrawImage(this.AvaloniaBitmap, sourceRect, destRect);
00076         }
00077 
00078         protected virtual void Dispose(bool disposing)
00079         {
00080             if (!disposedValue)
00081             {
00082                 if (disposing)
00083                 {
00084                     this.SKCanvas?.Dispose();
00085                     this.AvaloniaBitmap?.Dispose();
00086                 }
00087                 disposedValue = true;
00088             }
00089         }
00090 
00091         public void Dispose()
00092         {
00093             Dispose(disposing:  true);
00094             GC.SuppressFinalize(this);
00095         }
00096 
00097         public void Indispose()
00098         {
00099             if (!disposedValue)
00100             {
00101                 this.SKCanvas?.Dispose();
00102                 this.disposedValue = true;
00103             }
00104         }
00105     }
00106 }
```

8.5 SKMultiLayerRenderCanvas.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using Avalonia.Media.Imaging;
```

```
00021 using Avalonia.Skia;
00022 using SkiaSharp;
00023 using System;
00024 using System.Collections.Generic;
00025 using System.Linq;
00026 using System.Threading;
00027
00028 namespace VectSharp.Canvas
00029 {
00030     internal interface ISKRenderCanvas
00031     {
00032         void InvalidateDirty();
00033         void InvalidateZIndex();
00034     }
00035
00036 /// <summary>
00037 /// Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.
00038 /// </summary>
00039     public class SKMultiLayerRenderCanvas :  Avalonia.Controls.Control, IDisposable, ISKRenderCanvas
00040     {
00041 /// <summary>
00042 /// The width of the page that is rendered on this canvas.
00043 /// </summary>
00044         public double PageWidth { get; set; }
00045
00046 /// <summary>
00047 /// The height of the page that is rendered on this canvas.
00048 /// </summary>
00049         public double PageHeight { get; set; }
00050         private SKColor BackgroundColour;
00051
00052 /// <summary>
00053 /// Defines an overdraw margin for the clipping area.  Set this to a larger value if quickly moving
      the
00054 /// canvas around (e.g.  in a ZoomBorder) causes the edge of the image to disappear for a few
      milliseconds
00055 /// and that annoys you.  If <see cref="RelativePoint.Unit"/> is set to <see
      cref="RelativeUnit.Absolute"/>,
00056 /// the value corresponds to units on the <see cref="VectSharp.Page"/>; if it is set to <see
      cref="RelativeUnit.Relative"/>,
00057 /// it corresponds to a proportion of the visible area.
00058 /// </summary>
00059         public RelativePoint ClipMargin { get; set; } = new RelativePoint(0, 0,
      RelativeUnit.Absolute);
00060
00061 /// <summary>
00062 /// The list of render actions.  Each element in this list is itself a list, containing the actions
      that correspond to a layer in the image.
00063 /// </summary>
00064         public List<List<SKRenderAction>> RenderActions;
00065
00066 /// <summary>
00067 /// The list of transforms associated with each layer.
00068 /// </summary>
00069         public List<SKRenderAction> LayerTransforms;
00070
00071         private Dictionary<string, (SKBitmap, bool)> Images;
00072         private List<List<SKRenderAction>> TaggedRenderActions;
00073
00074 /// <summary>
00075 /// Create a new SKMultiLayerRenderCanvas from a <see cref="Document"/>, where each page represents a
      layer.
00076 /// </summary>
00077 /// <param name="document">The document containing the layers as <see cref="Page"/>s.</param>
00078 /// <param name="layerTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as the number of pages in <paramref name="document"/>.  This is
      useful to manipulate the position of each layer individually.  If this is null, an identity transform
      is applied to each layer.</param>
00079 /// <param name="backgroundColour">The background colour of the canvas.</param>
00080 /// <param name="width">The width of the canvas and the pages it contains.</param>
00081 /// <param name="height">The height of the canvas and the pages it contains.</param>
00082         public SKMultiLayerRenderCanvas(Document document, Colour backgroundColour, double width,
      double height, List<SKRenderAction> layerTransforms = null) :  this(document.Pages, backgroundColour,
      width, height, layerTransforms) { }
00083
00084 /// <summary>
00085 /// Create a new SKMultiLayerRenderCanvas from a collection of <see cref="Page"/>s, each representing
      a layer.
00086 /// </summary>
00087 /// <param name="layers">The contents of the canvas.  Each element in this list represents a
      layer.</param>
00088 /// <param name="layerTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as <paramref name="layers"/>.  This is useful to manipulate the
      position of each layer individually.  If this is null, an identity transform is applied to each
      layer.</param>
00089 /// <param name="backgroundColour">The background colour of the canvas.</param>
00090 /// <param name="width">The width of the canvas and the pages it contains.</param>
```

```
00091 /// <param name="height">The height of the canvas and the pages it contains.</param>
00092         public SKMultiLayerRenderCanvas(IEnumerable<Page> layers, Colour backgroundColour, double
      width, double height, List<SKRenderAction> layerTransforms = null)
00093         {
00094             List<SKRenderContext> contents = (from el in layers select
      el.CopyToSKRenderContext()).ToList();
00095             List<SKRenderAction> contentTransforms;
00096 
00097             if (layerTransforms == null)
00098             {
00099                 contentTransforms = (from el in Enumerable.Range(0, contents.Count) select
      SKRenderAction.TransformAction(SKMatrix.Identity)).ToList();
00100             }
00101             else
00102             {
00103                 contentTransforms = layerTransforms;
00104             }
00105 
00106             UpdateWith(contents, contentTransforms, backgroundColour, width, height);
00107 
00108             this.Width = width;
00109             this.Height = height;
00110 
00111             this.PointerPressed += this.PointerPressedAction;
00112             this.PointerReleased += this.PointerReleasedAction;
00113             this.PointerMoved += this.PointerMoveAction;
00114             this.PointerExited += this.PointerLeaveAction;
00115         }
00116 
00117 /// <summary>
00118 /// Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.
00119 /// </summary>
00120 /// <param name="contents">The contents of the canvas.  Each element in this list represents a layer.
      A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.</param>
00121 /// <param name="contentTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as <paramref name="contents"/>.  This is useful to manipulate the
      position of each layer individually.</param>
00122 /// <param name="backgroundColour">The background colour of the canvas.</param>
00123 /// <param name="width">The width of the canvas and the page it contains.</param>
00124 /// <param name="height">The height of the canvas and the page it contains.</param>
00125         public SKMultiLayerRenderCanvas(List<SKRenderContext> contents, List<SKRenderAction>
      contentTransforms, Colour backgroundColour, double width, double height)
00126         {
00127             UpdateWith(contents, contentTransforms, backgroundColour, width, height);
00128 
00129             this.Width = width;
00130             this.Height = height;
00131 
00132             this.PointerPressed += this.PointerPressedAction;
00133             this.PointerReleased += this.PointerReleasedAction;
00134             this.PointerMoved += this.PointerMoveAction;
00135             this.PointerExited += this.PointerLeaveAction;
00136         }
00137 
00138 /// <summary>
00139 /// Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.
00140 /// </summary>
00141 /// <param name="contents">The contents of the canvas.  Each element in this list represents a layer.
      A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.</param>
00142 /// <param name="contentTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as <paramref name="contents"/>.  This is useful to manipulate the
      position of each layer individually.</param>
00143 /// <param name="backgroundColour">The background colour of the canvas.</param>
00144 /// <param name="width">The width of the canvas and the page it contains.</param>
00145 /// <param name="height">The height of the canvas and the page it contains.</param>
00146         public void UpdateWith(List<SKRenderContext> contents, List<SKRenderAction> contentTransforms,
      Colour backgroundColour, double width, double height)
00147         {
00148             lock (RenderLock)
00149             {
00150                 if (this.RenderActions == null)
00151                 {
00152                     this.RenderActions = new List<List<SKRenderAction»();
00153                 }
00154                 else
00155                 {
00156                     this.RenderActions.Clear();
00157                 }
00158 
00159                 if (this.LayerTransforms == null)
00160                 {
00161                     this.LayerTransforms = new List<SKRenderAction>();
00162                 }
00163                 else
00164                 {
00165                     this.LayerTransforms.Clear();
00166                 }
```

```
00167
00168                 this.PageWidth = width;
00169                 this.PageHeight = height;
00170                 this.BackgroundColour = backgroundColour.ToSKColor();
00171
00172                 if (this.Images == null)
00173                 {
00174                     Images = new Dictionary<string, (SKBitmap, bool)>();
00175                 }
00176                 else
00177                 {
00178                     Images.Clear();
00179                 }
00180
00181                 for (int i = 0; i < contents.Count; i++)
00182                 {
00183                     this.LayerTransforms.Add(contentTransforms[i]);
00184                     this.RenderActions.Add(contents[i].SKRenderActions);
00185
00186                     foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in contents[i].Images)
00187                     {
00188                         this.Images[kvp.Key] = kvp.Value;
00189                     }
00190                 }
00191
00192                 if (this.TaggedRenderActions == null)
00193                 {
00194                     this.TaggedRenderActions = new List<List<SKRenderAction»();
00195                 }
00196                 else
00197                 {
00198                     this.TaggedRenderActions.Clear();
00199                 }
00200
00201
00202                 for (int i = this.RenderActions.Count - 1; i >= 0; i--)
00203                 {
00204                     TaggedRenderActions.Add(new List<SKRenderAction>());
00205
00206                     for (int j = this.RenderActions[i].Count - 1; j >= 0; j--)
00207                     {
00208                         RenderActions[i][j].InternalParent = this;
00209                         if (!string.IsNullOrEmpty(this.RenderActions[i][j].Tag))
00210                         {
00211                             TaggedRenderActions[this.RenderActions.Count - 1 -
    i].Add(this.RenderActions[i][j]);
00212                         }
00213                     }
00214                 }
00215             }
00216         }
00217
00218 /// <summary>
00219 /// Replace a single layer with the specified content.
00220 /// </summary>
00221 /// <param name="layer">The index of the layer to replace.</param>
00222 /// <param name="newContent">The new contents of the layer.  A Page can be converded to a
    SKRenderContext through the CopyToSKRenderContext method.</param>
00223 /// <param name="newTransform">The new transform for the layer.</param>
00224         public void UpdateLayer(int layer, SKRenderContext newContent, SKRenderAction newTransform)
00225         {
00226             lock (RenderLock)
00227             {
00228                 this.LayerTransforms[layer] = newTransform;
00229                 this.RenderActions[layer] = newContent.SKRenderActions;
00230
00231                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00232                 {
00233                     this.Images[kvp.Key] = kvp.Value;
00234                 }
00235
00236                 TaggedRenderActions[this.RenderActions.Count - 1 - layer].Clear();
00237
00238                 for (int j = this.RenderActions[layer].Count - 1; j >= 0; j--)
00239                 {
00240                     RenderActions[layer][j].InternalParent = this;
00241                     if (!string.IsNullOrEmpty(this.RenderActions[layer][j].Tag))
00242                     {
00243                         TaggedRenderActions[this.RenderActions.Count - 1 -
    layer].Add(this.RenderActions[layer][j]);
00244                     }
00245                 }
00246             }
00247
00248             this.InvalidateDirty();
00249         }
00250
```

```
00251 /// <summary>
00252 /// Add a new layer to the image.
00253 /// </summary>
00254 /// <param name="newContent">The contents of the new layer.  A Page can be converded to a
      SKRenderContext through the CopyToSKRenderContext method.</param>
00255 /// <param name="newTransform">The transform for the new layer.</param>
00256         public void AddLayer(SKRenderContext newContent, SKRenderAction newTransform)
00257         {
00258             lock (RenderLock)
00259             {
00260                 this.LayerTransforms.Add(newTransform);
00261                 this.RenderActions.Add(newContent.SKRenderActions);
00262 
00263                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00264                 {
00265                     this.Images[kvp.Key] = kvp.Value;
00266                 }
00267 
00268                 TaggedRenderActions.Insert(0, new List<SKRenderAction>());
00269 
00270                 for (int j = this.RenderActions[this.RenderActions.Count - 1].Count - 1; j >= 0; j--)
00271                 {
00272                     RenderActions[this.RenderActions.Count - 1][j].InternalParent = this;
00273                     if (!string.IsNullOrEmpty(this.RenderActions[this.RenderActions.Count -
      1][j].Tag))
00274                     {
00275                         TaggedRenderActions[0].Add(this.RenderActions[this.RenderActions.Count -
      1][j]);
00276                     }
00277                 }
00278             }
00279 
00280             this.InvalidateDirty();
00281         }
00282 
00283 /// <summary>
00284 /// Insert a new layer at the specified index.
00285 /// </summary>
00286 /// <param name="index">The position at which the new layer will be inserted.</param>
00287 /// <param name="newContent">The contents of the new layer.</param>
00288 /// <param name="newTransform">The transform for the new layer.</param>
00289         public void InsertLayer(int index, SKRenderContext newContent, SKRenderAction newTransform)
00290         {
00291             lock (RenderLock)
00292             {
00293                 this.LayerTransforms.Insert(index, newTransform);
00294                 this.RenderActions.Insert(index, newContent.SKRenderActions);
00295 
00296                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00297                 {
00298                     this.Images[kvp.Key] = kvp.Value;
00299                 }
00300 
00301                 TaggedRenderActions.Insert(this.RenderActions.Count - 1 - index, new
      List<SKRenderAction>());
00302 
00303                 for (int j = this.RenderActions[index].Count - 1; j >= 0; j--)
00304                 {
00305                     RenderActions[index][j].InternalParent = this;
00306                     if (!string.IsNullOrEmpty(this.RenderActions[index][j].Tag))
00307                     {
00308                         TaggedRenderActions[this.RenderActions.Count - 1 -
      index].Add(this.RenderActions[index][j]);
00309                     }
00310                 }
00311             }
00312 
00313             this.InvalidateDirty();
00314         }
00315 
00316 /// <summary>
00317 /// Remove the specified layer from the image.
00318 /// </summary>
00319 /// <param name="layer">The index of the layer to remove.</param>
00320         public void RemoveLayer(int layer)
00321         {
00322             lock (RenderLock)
00323             {
00324                 this.LayerTransforms[layer].Dispose();
00325                 this.LayerTransforms.RemoveAt(layer);
00326 
00327                 for (int i = 0; i < this.RenderActions[layer].Count; i++)
00328                 {
00329                     this.RenderActions[layer][i].Dispose();
00330                 }
00331                 this.RenderActions.RemoveAt(layer);
00332 
```

```
00333                 TaggedRenderActions.RemoveAt(this.TaggedRenderActions.Count - 1 - layer);
00334             }
00335 
00336             this.InvalidateDirty();
00337         }
00338 
00339 /// <summary>
00340 /// Switch the position of the two specified layers.
00341 /// </summary>
00342 /// <param name="layer1">The index of the first layer to switch.</param>
00343 /// <param name="layer2">The index of the second layer to switch.</param>
00344         public void SwitchLayers(int layer1, int layer2)
00345         {
00346             lock (RenderLock)
00347             {
00348                 var temp = this.LayerTransforms[layer1];
00349                 this.LayerTransforms[layer1] = this.LayerTransforms[layer2];
00350                 this.LayerTransforms[layer2] = temp;
00351 
00352                 var temp2 = this.RenderActions[layer1];
00353                 this.RenderActions[layer1] = this.RenderActions[layer2];
00354                 this.RenderActions[layer2] = temp2;
00355 
00356                 var temp3 = this.TaggedRenderActions[this.RenderActions.Count - 1 - layer1];
00357                 this.TaggedRenderActions[this.RenderActions.Count - 1 - layer1] =
    this.TaggedRenderActions[this.RenderActions.Count - 1 - layer2];
00358                 this.TaggedRenderActions[this.RenderActions.Count - 1 - layer2] = temp3;
00359             }
00360 
00361             this.InvalidateDirty();
00362         }
00363 
00364 /// <summary>
00365 /// Move the specified layer to the specified position, shifting all other layers as necessary.
00366 /// </summary>
00367 /// <param name="oldIndex">The current index of the layer to move.</param>
00368 /// <param name="newIndex">The final index of the layer.  Layers after this will be shifted by 1 in
    order to accommodate the moved layer.</param>
00369         public void MoveLayer(int oldIndex, int newIndex)
00370         {
00371             lock (RenderLock)
00372             {
00373                 var temp = this.LayerTransforms[oldIndex];
00374                 this.LayerTransforms.RemoveAt(oldIndex);
00375                 this.LayerTransforms.Insert(newIndex, temp);
00376 
00377                 var temp2 = this.RenderActions[oldIndex];
00378                 this.RenderActions.RemoveAt(oldIndex);
00379                 this.RenderActions.Insert(newIndex, temp2);
00380 
00381                 var temp3 = this.TaggedRenderActions[this.RenderActions.Count - 1 - oldIndex];
00382                 this.TaggedRenderActions.RemoveAt(this.RenderActions.Count - 1 - oldIndex);
00383                 this.TaggedRenderActions.Insert(this.RenderActions.Count - 1 - newIndex, temp3);
00384             }
00385 
00386             this.InvalidateDirty();
00387         }
00388 
00389 
00390         private SKRenderAction CurrentPressedAction = null;
00391         private void PointerPressedAction(object sender, Avalonia.Input.PointerPressedEventArgs e)
00392         {
00393             SKPoint position = e.GetPosition(this).ToSKPoint();
00394 
00395             for (int i = 0; i < TaggedRenderActions.Count; i++)
00396             {
00397                 bool found = false;
00398 
00399                 for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00400                 {
00401                     if (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X,
    position.Y) == true)
00402                     {
00403 
00404                         if (TaggedRenderActions[i][j].ActionType == SKRenderAction.ActionTypes.Path)
00405                         {
00406                             CurrentPressedAction = TaggedRenderActions[i][j];
00407                             TaggedRenderActions[i][j].FirePointerPressed(e);
00408                             found = true;
00409                             break;
00410                         }
00411                         else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.Text)
00412                         {
00413                             CurrentPressedAction = TaggedRenderActions[i][j];
00414                             TaggedRenderActions[i][j].FirePointerPressed(e);
00415                             found = true;
```

```
00416                           break;
00417                       }
00418                       else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.RasterImage)
00419                       {
00420                           CurrentPressedAction = TaggedRenderActions[i][j];
00421                           TaggedRenderActions[i][j].FirePointerPressed(e);
00422                           found = true;
00423                           break;
00424                       }
00425                   }
00426               }
00427
00428               if (found)
00429               {
00430                   break;
00431               }
00432           }
00433       }
00434
00435       private void PointerReleasedAction(object sender, Avalonia.Input.PointerReleasedEventArgs e)
00436       {
00437           if (CurrentPressedAction != null)
00438           {
00439               if (!CurrentPressedAction.Disposed)
00440               {
00441                   CurrentPressedAction.FirePointerReleased(e);
00442               }
00443               CurrentPressedAction = null;
00444           }
00445           else
00446           {
00447               SKPoint position = e.GetPosition(this).ToSKPoint();
00448
00449               for (int i = 0; i < TaggedRenderActions.Count; i++)
00450               {
00451                   bool found = false;
00452
00453                   for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00454                   {
00455                       if
    (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X, position.Y) == true)
00456                       {
00457                           if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.Path)
00458                           {
00459                               TaggedRenderActions[i][j].FirePointerReleased(e);
00460                               found = true;
00461                               break;
00462                           }
00463                           else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.Text)
00464                           {
00465                               TaggedRenderActions[i][j].FirePointerReleased(e);
00466                               found = true;
00467                               break;
00468                           }
00469                           else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.RasterImage)
00470                           {
00471                               TaggedRenderActions[i][j].FirePointerReleased(e);
00472                               found = true;
00473                               break;
00474                           }
00475                       }
00476                   }
00477
00478                   if (found)
00479                   {
00480                       break;
00481                   }
00482               }
00483           }
00484       }
00485
00486       private SKRenderAction CurrentOverAction = null;
00487       private void PointerMoveAction(object sender, Avalonia.Input.PointerEventArgs e)
00488       {
00489           SKPoint position = e.GetPosition(this).ToSKPoint();
00490
00491           bool found = false;
00492
00493           for (int i = 0; i < TaggedRenderActions.Count; i++)
00494           {
00495               for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00496               {
00497                   if (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X,
```

```
    position.Y) == true)
00498                   {
00499                       if (TaggedRenderActions[i][j].ActionType == SKRenderAction.ActionTypes.Path)
00500                       {
00501                           found = true;
00502
00503                           if (CurrentOverAction != TaggedRenderActions[i][j])
00504                           {
00505                               if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00506                               {
00507                                   CurrentOverAction.FirePointerExited(e);
00508                               }
00509                               CurrentOverAction = TaggedRenderActions[i][j];
00510                               CurrentOverAction.FirePointerEntered(e);
00511                           }
00512
00513                           break;
00514                       }
00515                       else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.Text)
00516                       {
00517                           found = true;
00518
00519                           if (CurrentOverAction != TaggedRenderActions[i][j])
00520                           {
00521                               if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00522                               {
00523                                   CurrentOverAction.FirePointerExited(e);
00524                               }
00525                               CurrentOverAction = TaggedRenderActions[i][j];
00526                               CurrentOverAction.FirePointerEntered(e);
00527                           }
00528
00529                           break;
00530                       }
00531                       else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.RasterImage)
00532                       {
00533                           found = true;
00534
00535                           if (CurrentOverAction != TaggedRenderActions[i][j])
00536                           {
00537                               if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00538                               {
00539                                   CurrentOverAction.FirePointerExited(e);
00540                               }
00541                               CurrentOverAction = TaggedRenderActions[i][j];
00542                               CurrentOverAction.FirePointerEntered(e);
00543                           }
00544
00545                           break;
00546                       }
00547                   }
00548               }
00549
00550               if (found)
00551               {
00552                   break;
00553               }
00554           }
00555
00556           if (!found)
00557           {
00558               if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00559               {
00560                   CurrentOverAction.FirePointerExited(e);
00561               }
00562               CurrentOverAction = null;
00563           }
00564       }
00565
00566       private void PointerLeaveAction(object sender, Avalonia.Input.PointerEventArgs e)
00567       {
00568           if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00569           {
00570               CurrentOverAction.FirePointerExited(e);
00571           }
00572           CurrentOverAction = null;
00573       }
00574
00575 /// <inheritdoc/>
00576       protected override void OnAttachedToVisualTree(VisualTreeAttachmentEventArgs e)
00577       {
00578           base.OnAttachedToVisualTree(e);
00579
00580           StartRenderingThread();
00581       }
```

```
00582
00583 /// <inheritdoc/>
00584         protected override void OnDetachedFromVisualTree(VisualTreeAttachmentEventArgs e)
00585         {
00586             StopRenderingThread();
00587         }
00588
00589         private void StopRenderingThread()
00590         {
00591             DisposedHandle.Set();
00592         }
00593
00594         private EventWaitHandle DisposedHandle;
00595         private EventWaitHandle RenderRequestedHandle;
00596
00597         private RenderingParameters RenderingRequest = null;
00598         private readonly object RenderingRequestLock = new object();
00599
00600         private void StartRenderingThread()
00601         {
00602             DisposedHandle = new EventWaitHandle(false, EventResetMode.ManualReset);
00603             RenderRequestedHandle = new EventWaitHandle(false, EventResetMode.ManualReset);
00604
00605             Thread renderingThread = new Thread(async () =>
00606             {
00607                 bool finished = false;
00608                 WaitHandle[] handles = new WaitHandle[] { DisposedHandle, RenderRequestedHandle };
00609
00610                 while (!finished)
00611                 {
00612                     int handle = EventWaitHandle.WaitAny(handles);
00613
00614                     if (handle == 0)
00615                     {
00616                         finished = true;
00617                     }
00618                     else if (handle == 1)
00619                     {
00620                         RenderingParameters requestParams;
00621
00622                         lock (RenderingRequestLock)
00623                         {
00624                             requestParams = RenderingRequest.Clone();
00625                             RenderRequestedHandle.Reset();
00626                         }
00627
00628                         SKCanvas canvas;
00629
00630                         if (BackBuffer == null || BackBufferRenderingParams == null ||
      requestParams.RenderWidth > BackBufferRenderingParams.RenderWidth || requestParams.RenderHeight >
      BackBufferRenderingParams.RenderHeight)
00631                         {
00632                             await Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00633                             {
00634                                 SkiaBitmap tempBufferReference = BackBuffer;
00635
00636                                 _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00637                                 {
00638                                     tempBufferReference?.Dispose();
00639                                 }, Avalonia.Threading.DispatcherPriority.Send);
00640
00641                                 BackBuffer = new SkiaBitmap(requestParams.RenderWidth,
      requestParams.RenderHeight);
00642                                 BackBufferSkiaCanvas = BackBuffer.SKCanvas;
00643                             }, Avalonia.Threading.DispatcherPriority.MaxValue);
00644                         }
00645
00646                         canvas = BackBufferSkiaCanvas;
00647
00648                         canvas.Save();
00649
00650                         canvas.Scale(requestParams.Scale);
00651                         canvas.Translate(-requestParams.Left, -requestParams.Top);
00652
00653                         canvas.Clear(BackgroundColour);
00654
00655                         canvas.ClipRect(new SKRect(requestParams.Left, requestParams.Top,
      requestParams.Left + requestParams.Width, requestParams.Top + requestParams.Height));
00656
00657                         RenderImage(canvas);
00658
00659                         canvas.RestoreToCount(-1);
00660
00661                         BackBufferRenderingParams = requestParams;
00662
00663                         lock (FrontBufferLock)
00664                         {
```

```
00665                           SkiaBitmap tempFrontReference = FrontBuffer;
00666                           RenderingParameters tempFrontRenderingParameters =
    FrontBufferRenderingParams;
00667                           SKCanvas tempFrontCanvas = FrontBufferSkiaCanvas;
00668 
00669                           FrontBuffer = BackBuffer;
00670                           FrontBufferRenderingParams = BackBufferRenderingParams;
00671                           FrontBufferSkiaCanvas = BackBufferSkiaCanvas;
00672 
00673                           BackBuffer = BackBuffer2;
00674                           BackBufferRenderingParams = BackBufferRenderingParams2;
00675                           BackBufferSkiaCanvas = BackBufferSkiaCanvas2;
00676 
00677                           BackBuffer2 = tempFrontReference;
00678                           BackBufferRenderingParams2 = tempFrontRenderingParameters;
00679                           BackBufferSkiaCanvas2 = tempFrontCanvas;
00680                       }
00681 
00682                       await Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00683                       {
00684                           base.InvalidateVisual();
00685                       }, Avalonia.Threading.DispatcherPriority.MaxValue);
00686                   }
00687               }
00688           });
00689 
00690           renderingThread.Start();
00691       }
00692 
00693 /// <summary>
00694 /// A lock for the rendering loop.  The public methods of this class already lock on this, but you may
    need it if you want to directly manipulate the contents of the canvas.
00695 /// </summary>
00696       public object RenderLock = new object();
00697 
00698 /// <summary>
00699 /// Render the image to a bitmap at the specified resolution.
00700 /// </summary>
00701 /// <param name="width">The width of the rendered image.  Note that the actual width of the returned
    image might be lower than this, depending on the aspect ratio of the image.</param>
00702 /// <param name="height">The height of the rendered image.  Note that the actual height of the
    returned image might be lower than this, depending on the aspect ratio of the image.</param>
00703 /// <param name="background">The background colour for the image.  If this is <see langword="null" />,
    the current background colour is used.</param>
00704 /// <returns>A <see cref="RenderTargetBitmap"/> containing the image rendered at the specified
    resolution.</returns>
00705       public RenderTargetBitmap RenderAtResolution(int width, int height, SKColor?  background =
    null)
00706       {
00707           SKColor realBackground = background ??  this.BackgroundColour;
00708 
00709           double scale = Math.Min(width / this.PageWidth, height / this.PageHeight);
00710 
00711           width = (int)Math.Round(this.PageWidth * scale);
00712           height = (int)Math.Round(this.PageHeight * scale);
00713 
00714           SkiaBitmap skBmp = new SkiaBitmap(width, height);
00715           SKCanvas canvas = skBmp.SKCanvas;
00716 
00717           canvas.Clear(realBackground);
00718 
00719           canvas.Save();
00720 
00721           canvas.Scale((float)scale);
00722 
00723           RenderImage(canvas);
00724 
00725           canvas.RestoreToCount(-1);
00726 
00727           skBmp.Indispose();
00728 
00729           return skBmp.AvaloniaBitmap;
00730       }
00731 
00732       private void RenderImage(SKCanvas canvas)
00733       {
00734           lock (RenderLock)
00735           {
00736               for (int i = 0; i < this.RenderActions.Count; i++)
00737               {
00738                   canvas.Save();
00739 
00740                   if (this.LayerTransforms[i].ActionType == SKRenderAction.ActionTypes.Transform)
00741                   {
00742                       SKMatrix mat = this.LayerTransforms[i].Transform.Value;
00743                       canvas.Concat(ref mat);
00744                   }
```

```
00745
00746                     RenderLayer(canvas, i);
00747                     canvas.Restore();
00748                 }
00749             }
00750         }
00751
00752         private void RenderLayer(SKCanvas canvas, int layer)
00753         {
00754             bool updateHitTests = this.TaggedRenderActions[this.TaggedRenderActions.Count - 1 -
      layer].Count > 0;
00755
00756             HashSet<uint> ZIndices = new HashSet<uint>();
00757
00758             canvas.Save();
00759
00760             SKMatrix invertedInitialTransform;
00761
00762             if (this.LayerTransforms[layer].ActionType == SKRenderAction.ActionTypes.Transform)
00763             {
00764                 invertedInitialTransform =
      (canvas.TotalMatrix.PreConcat(this.LayerTransforms[layer].Transform.Value.Invert())).Invert();
00765             }
00766             else
00767             {
00768                 invertedInitialTransform = canvas.TotalMatrix.Invert();
00769             }
00770
00771             SKPath clipPath = null;
00772             Stack<SKPath> clipPaths = new Stack<SKPath>();
00773             clipPaths.Push(null);
00774
00775             for (int i = 0; i < this.RenderActions[layer].Count; i++)
00776             {
00777                 ZIndices.Add(this.RenderActions[layer][i].ZIndex);
00778
00779                 if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Clip)
00780                 {
00781                     canvas.ClipPath(this.RenderActions[layer][i].Path, antialias:  true);
00782
00783                     if (updateHitTests)
00784                     {
00785                         if (clipPath == null)
00786                         {
00787                             clipPath = this.RenderActions[layer][i].Path.Clone();
00788
      clipPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00789                         }
00790                         else
00791                         {
00792                             using (SKPath tempPath = this.RenderActions[layer][i].Path.Clone())
00793                             {
00794
      tempPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00795                                 clipPath.Op(tempPath, SKPathOp.Intersect);
00796                             }
00797                         }
00798                     }
00799                 }
00800                 else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Restore)
00801                 {
00802                     canvas.Restore();
00803
00804                     if (updateHitTests)
00805                     {
00806                         clipPath = clipPaths.Pop();
00807                     }
00808                 }
00809                 else if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Save)
00810                 {
00811                     canvas.Save();
00812
00813                     if (updateHitTests)
00814                     {
00815                         clipPaths.Push(clipPath?.Clone());
00816                     }
00817                 }
00818                 else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Transform)
00819                 {
00820                     SKMatrix mat = this.RenderActions[layer][i].Transform.Value;
00821                     canvas.Concat(ref mat);
00822                 }
00823                 else
00824                 {
00825                     if (this.RenderActions[layer][i].ZIndex == 0)
```

```
00826                     {
00827
00828                         if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Path
      && this.RenderActions[layer][i].Path != null)
00829                         {
00830                             canvas.DrawPath(this.RenderActions[layer][i].Path,
      this.RenderActions[layer][i].Paint);
00831                         }
00832                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Text)
00833                         {
00834                             canvas.DrawText(this.RenderActions[layer][i].Text,
      this.RenderActions[layer][i].TextX, this.RenderActions[layer][i].TextY,
      this.RenderActions[layer][i].Font, this.RenderActions[layer][i].Paint);
00835                         }
00836                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00837                         {
00838                             (SKBitmap image, bool interpolate) =
      this.Images[this.RenderActions[layer][i].ImageId];
00839
00840                             SKPaint paint;
00841
00842                             if (!interpolate)
00843                             {
00844                                 paint = null;
00845                             }
00846                             else
00847                             {
00848                                 paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
00849                             }
00850
00851                             canvas.DrawBitmap(image, this.RenderActions[layer][i].ImageSource.Value,
      this.RenderActions[layer][i].ImageDestination.Value, paint);
00852
00853                             paint?.Dispose();
00854                         }
00855                     }
00856
00857                     if (updateHitTests && this.RenderActions[layer][i].Tag != null &&
      this.RenderActions[layer][i].HitTestPath != null)
00858                     {
00859                         SKPath hitTestPath = this.RenderActions[layer][i].HitTestPath.Clone();
00860
      hitTestPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00861
00862                         if (clipPath != null)
00863                         {
00864                             hitTestPath.Op(clipPath, SKPathOp.Intersect);
00865                         }
00866
00867                         this.RenderActions[layer][i].LastRenderedGlobalHitTestPath = hitTestPath;
00868                     }
00869                 }
00870             }
00871
00872             canvas.Restore();
00873
00874             uint[] sortedIndices = ZIndices.OrderBy(x => x).ToArray();
00875
00876             if (sortedIndices.Length > 1 || sortedIndices[0] != 0)
00877             {
00878                 for (int j = 0; j < sortedIndices.Length; j++)
00879                 {
00880                     canvas.Save();
00881
00882                     for (int i = 0; i < this.RenderActions[layer].Count; i++)
00883                     {
00884                         if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Clip)
00885                         {
00886                             canvas.ClipPath(this.RenderActions[layer][i].Path, antialias:  true);
00887                         }
00888                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Restore)
00889                         {
00890                             canvas.Restore();
00891                         }
00892                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Save)
00893                         {
00894                             canvas.Save();
00895                         }
00896                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Transform)
00897                         {
00898                             SKMatrix mat = this.RenderActions[layer][i].Transform.Value;
```

```
00899                               canvas.Concat(ref mat);
00900                           }
00901                           else if (sortedIndices[j] != 0 && this.RenderActions[layer][i].ZIndex ==
      sortedIndices[j])
00902                           {
00903                               if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Path && this.RenderActions[layer][i].Path != null)
00904                               {
00905                                   canvas.DrawPath(this.RenderActions[layer][i].Path,
      this.RenderActions[layer][i].Paint);
00906                               }
00907                               else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Text)
00908                               {
00909                                   canvas.DrawText(this.RenderActions[layer][i].Text,
      this.RenderActions[layer][i].TextX, this.RenderActions[layer][i].TextY,
      this.RenderActions[layer][i].Font, this.RenderActions[layer][i].Paint);
00910                               }
00911                               else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00912                               {
00913                                   (SKBitmap image, bool interpolate) =
      this.Images[this.RenderActions[layer][i].ImageId];
00914
00915                                   SKPaint paint;
00916
00917                                   if (!interpolate)
00918                                   {
00919                                       paint = null;
00920                                   }
00921                                   else
00922                                   {
00923                                       paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
00924                                   }
00925
00926                                   canvas.DrawBitmap(image,
      this.RenderActions[layer][i].ImageSource.Value, this.RenderActions[layer][i].ImageDestination.Value,
      paint);
00927
00928                                   paint?.Dispose();
00929                               }
00930                           }
00931                       }
00932
00933                       canvas.Restore();
00934                   }
00935               }
00936           }
00937
00938           private bool IsDirty = false;
00939
00940 /// <summary>
00941 /// Invalidate the contents of the canvas, forcing it to redraw itself.
00942 /// </summary>
00943           public void InvalidateDirty()
00944           {
00945               this.IsDirty = true;
00946               base.InvalidateVisual();
00947           }
00948
00949 /// <summary>
00950 /// Invalidate the contents of the canvas, specifying that the order of the layers has changed.
00951 /// </summary>
00952           public void InvalidateZIndex()
00953           {
00954               for (int i = 0; i < this.TaggedRenderActions.Count; i++)
00955               {
00956                   this.TaggedRenderActions[i] = this.TaggedRenderActions[i].OrderByDescending(a =>
      a.ZIndex).ToList();
00957               }
00958               this.InvalidateDirty();
00959           }
00960
00961
00962           private SkiaBitmap FrontBuffer = null;
00963           private RenderingParameters FrontBufferRenderingParams = null;
00964           private readonly object EmptyFrontBufferLock = new object();
00965           private object FrontBufferLock => FrontBuffer?.Lock ??  EmptyFrontBufferLock;
00966           SKCanvas FrontBufferSkiaCanvas = null;
00967
00968           private SkiaBitmap BackBuffer = null;
00969           private RenderingParameters BackBufferRenderingParams = null;
00970           SKCanvas BackBufferSkiaCanvas = null;
00971
00972           private SkiaBitmap BackBuffer2 = null;
00973           private RenderingParameters BackBufferRenderingParams2 = null;
00974           SKCanvas BackBufferSkiaCanvas2 = null;
```

```
00975
00976 /// <inheritdoc/>
00977         public override void Render(DrawingContext context)
00978         {
00979             lock (FrontBufferLock)
00980             {
00981                 double scale;
00982
00983                 PixelPoint layoutTopLeft = this.PointToScreen(new Avalonia.Point(0, 0));
00984                 PixelPoint layoutBottomRight = this.PointToScreen(new
      Avalonia.Point(this.Bounds.Width, this.Bounds.Height));
00985
00986                 scale = 0.5 * (this.PageWidth / (layoutBottomRight.X - layoutTopLeft.X) +
      this.PageHeight / (layoutBottomRight.Y - layoutTopLeft.Y));
00987
00988                 Avalonia.Controls.Control parent = this.Parent as Avalonia.Controls.Control;
00989
00990                 double left;
00991                 double top;
00992                 double width;
00993                 double height;
00994
00995                 if (this.ClipToBounds && parent != null)
00996                 {
00997                     Avalonia.Point clipTopLeft = this.PointToClient(parent.PointToScreen(new
      Avalonia.Point(0, 0)));
00998                     Avalonia.Point clipBottomRight = this.PointToClient(parent.PointToScreen(new
      Avalonia.Point(parent.Bounds.Width, parent.Bounds.Height)));
00999
01000                     Rect clipBounds;
01001
01002                     if (ClipMargin.Unit == RelativeUnit.Absolute)
01003                     {
01004                         clipBounds = new Rect(clipTopLeft.X - ClipMargin.Point.X, clipTopLeft.Y -
      ClipMargin.Point.Y, clipBottomRight.X - clipTopLeft.X + ClipMargin.Point.X * 2, clipBottomRight.Y -
      clipTopLeft.Y + ClipMargin.Point.Y * 2);
01005                     }
01006                     else
01007                     {
01008                         double clipW = clipBottomRight.X - clipTopLeft.X;
01009                         double clipH = clipBottomRight.Y - clipTopLeft.Y;
01010
01011                         clipBounds = new Rect(clipTopLeft.X - clipW * ClipMargin.Point.X,
      clipTopLeft.Y - clipH * ClipMargin.Point.Y, clipW * (1 + 2 * ClipMargin.Point.X), clipH * (1 + 2 *
      ClipMargin.Point.Y));
01012                     }
01013
01014                     left = Math.Max(0, clipBounds.Left);
01015                     top = Math.Max(0, clipBounds.Top);
01016
01017                     width = Math.Min(clipBounds.Width, this.PageWidth);
01018                     height = Math.Min(clipBounds.Height, this.PageHeight);
01019                 }
01020                 else
01021                 {
01022                     left = 0;
01023                     top = 0;
01024
01025                     width = this.PageWidth;
01026                     height = this.PageHeight;
01027                 }
01028
01029                 int pixelWidth = (int)Math.Round(width / scale);
01030                 int pixelHeight = (int)Math.Round(height / scale);
01031
01032                 if (pixelWidth > 0 && pixelHeight > 0)
01033                 {
01034                     if (!this.ClipToBounds && (pixelWidth * pixelHeight >= int.MaxValue / 8 ||
      pixelWidth * pixelHeight < 0))
01035                     {
01036                         double ratio = width / height;
01037
01038                         pixelHeight = (int)Math.Round(Math.Sqrt(int.MaxValue / (8 * ratio)));
01039                         pixelWidth = (int)Math.Round(pixelHeight * ratio);
01040                     }
01041
01042                     PixelPoint topLeftScreen = this.PointToScreen(new Avalonia.Point(left, top));
01043                     Avalonia.Point topLeft = this.PointToClient(topLeftScreen);
01044                     Avalonia.Point bottomRight = this.PointToClient(new PixelPoint(topLeftScreen.X +
      pixelWidth, topLeftScreen.Y + pixelHeight));
01045
01046                     Rect targetRect = new Rect(topLeft, bottomRight);
01047
01048                     pixelWidth = this.PointToScreen(targetRect.BottomRight).X -
      this.PointToScreen(targetRect.TopLeft).X;
01049                     pixelHeight = this.PointToScreen(targetRect.BottomRight).Y -
      this.PointToScreen(targetRect.TopLeft).Y;
```

```
01050
01051                     RenderingParameters currentParameters = new RenderingParameters((float)left,
      (float)top, (float)width, (float)height, (float)(1 / scale), pixelWidth, pixelHeight);
01052
01053                     if (FrontBuffer != null && FrontBufferRenderingParams == currentParameters &&
      !IsDirty)
01054                     {
01055                         if (RenderRequestedHandle.WaitOne(0))
01056                         {
01057                             Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01058                             {
01059                                 this.InvalidateVisual();
01060                             });
01061                         }
01062
01063                         context.DrawImage(FrontBuffer, new Rect(0, 0, pixelWidth, pixelHeight),
      targetRect);
01064                     }
01065                     else if (FrontBuffer != null &&
      FrontBufferRenderingParams.GoodEnough(currentParameters) && !IsDirty)
01066                     {
01067                         if (RenderRequestedHandle.WaitOne(0))
01068                         {
01069                             Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01070                             {
01071                                 this.InvalidateVisual();
01072                             });
01073                         }
01074                         context.DrawImage(FrontBuffer, new Rect(0, 0,
      FrontBufferRenderingParams.RenderWidth, FrontBufferRenderingParams.RenderHeight), new
      Rect(FrontBufferRenderingParams.Left, FrontBufferRenderingParams.Top,
      FrontBufferRenderingParams.Width, FrontBufferRenderingParams.Height));
01075                     }
01076                     else
01077                     {
01078                         lock (RenderingRequestLock)
01079                         {
01080                             IsDirty = false;
01081                             RenderingRequest = currentParameters;
01082                             RenderRequestedHandle.Set();
01083                         }
01084
01085                         if (FrontBuffer != null)
01086                         {
01087                             context.DrawImage(FrontBuffer, new Rect(0, 0,
      FrontBufferRenderingParams.RenderWidth, FrontBufferRenderingParams.RenderHeight), new
      Rect(FrontBufferRenderingParams.Left, FrontBufferRenderingParams.Top,
      FrontBufferRenderingParams.Width, FrontBufferRenderingParams.Height));
01088                         }
01089
01090                         Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01091                         {
01092                             this.InvalidateVisual();
01093                         });
01094                     }
01095                 }
01096             }
01097         }
01098
01099         private bool disposedValue;
01100
01101 /// <inheritdoc cref="Dispose()"/>
01102         protected virtual void Dispose(bool disposing)
01103         {
01104             if (!disposedValue)
01105             {
01106                 if (disposing)
01107                 {
01108                     this.BackBuffer?.Dispose();
01109                     this.BackBuffer2?.Dispose();
01110                     this.DisposedHandle?.Set();
01111                     this.FrontBuffer?.Dispose();
01112
01113                     foreach (KeyValuePair<string, (SKBitmap, bool)> image in this.Images)
01114                     {
01115                         image.Value.Item1?.Dispose();
01116                     }
01117
01118                     for (int i = 0; i < this.RenderActions?.Count; i++)
01119                     {
01120                         foreach (SKRenderAction act in this.RenderActions[i])
01121                         {
01122                             act?.Dispose();
01123                         }
01124
01125                         this.LayerTransforms?[i]?.Dispose();
01126                     }
```

```
01127                 }
01128
01129                 disposedValue = true;
01130             }
01131         }
01132
01133 /// <inheritdoc/>
01134         public void Dispose()
01135         {
01136             Dispose(disposing:  true);
01137             GC.SuppressFinalize(this);
01138         }
01139     }
01140 }
```

8.6 SKRenderContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using SkiaSharp;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 using VectSharp.Filters;
00023
00024 namespace VectSharp.Canvas
00025 {
00026
00027 /// <summary>
00028 /// Represents a light-weight rendering action.
00029 /// </summary>
00030     public class SKRenderAction :  IDisposable
00031     {
00032 /// <summary>
00033 /// Returns a boolean value indicating whether the current instance has been disposed.
00034 /// </summary>
00035         public bool Disposed => disposedValue;
00036
00037         private bool disposedValue;
00038
00039 /// <summary>
00040 /// Types of rendering actions.
00041 /// </summary>
00042         public enum ActionTypes
00043         {
00044 /// <summary>
00045 /// The render action represents a path object.
00046 /// </summary>
00047             Path,
00048
00049 /// <summary>
00050 /// The render action represents a text object.
00051 /// </summary>
00052             Text,
00053
00054 /// <summary>
00055 /// The render action represents a raster image.
00056 /// </summary>
00057             RasterImage,
00058
00059 /// <summary>
00060 /// The render action represents a transformation of the coordinate space.
00061 /// </summary>
00062             Transform,
00063
00064 /// <summary>
00065 /// The render action represents saving the current graphics state.
00066 /// </summary>
```

```
00067             Save,
00068
00069 /// <summary>
00070 /// The render action represents restoring the last saved graphics state.
00071 /// </summary>
00072             Restore,
00073
00074 /// <summary>
00075 /// The render action represents an update of the current clip path.
00076 /// </summary>
00077             Clip,
00078
00079 /// <summary>
00080 /// The render action represents rendering a graphics object with a filter.
00081 /// </summary>
00082             DrawFiltered
00083         }
00084
00085 /// <summary>
00086 /// Type of the rendering action.
00087 /// </summary>
00088         public ActionTypes ActionType { get; private set; }
00089
00090 /// <summary>
00091 /// Path that needs to be rendered (null if the action type is not <see cref="ActionTypes.Path"/>).
      If you change this, you probably want to call this object's <see cref="InvalidateHitTestPath"/>
      method.
00092 /// </summary>
00093         public SKPath Path { get; set; }
00094
00095         internal SKPath HitTestPath { get; set; }
00096         internal SKPath LastRenderedGlobalHitTestPath { get; set; }
00097
00098 /// <summary>
00099 /// Text that needs to be rendered (null if the action type is not <see cref="ActionTypes.Text"/>).
      If you change this, you probably want to call this object's <see cref="InvalidateHitTestPath"/>
      method.
00100 /// </summary>
00101         public string Text { get; set; }
00102
00103 /// <summary>
00104 /// The font that will be used to render the text (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00105 /// </summary>
00106         public SKFont Font { get; set; }
00107
00108 /// <summary>
00109 /// The X coordainate at which the text will be drawn (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00110 /// </summary>
00111         public float TextX { get; set; }
00112
00113 /// <summary>
00114 /// The Y coordainate at which the text will be drawn (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00115 /// </summary>
00116         public float TextY { get; set; }
00117
00118 /// <summary>
00119 /// Paint used to render the text or path (<see langword="null"/> if the action type is neither <see
      cref="ActionTypes.Text"/> nor <see cref="ActionTypes.Path"/>).  If you change this, you probably want
      to call this object's <see cref="InvalidateHitTestPath"/> method.
00120 /// </summary>
00121         public SKPaint Paint { get; set; }
00122
00123 /// <summary>
00124 /// Univocal identifier of the image that needs to be drawn.
00125 /// </summary>
00126         public string ImageId { get; set; }
00127
00128 /// <summary>
00129 /// The source rectangle of the image (<see langword="null"/> if the action type is not <see
      cref="ActionTypes.RasterImage"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateVisual"/> method.
00130 /// </summary>
00131         public SKRect?  ImageSource { get; set; }
00132
00133 /// <summary>
00134 /// The destination rectangle of the image (<see langword="null"/> if the action type is not <see
      cref="ActionTypes.RasterImage"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00135 /// </summary>
00136         public SKRect?  ImageDestination { get; set; }
00137
```

```
00138 /// <summary>
00139 /// The transformation matrix that will be applied to the current coordinate system (<see
      langword="null"/> if the action type is not <see cref="ActionTypes.Transform"/>).  If you change this,
      you probably want to call this object's <see cref="InvalidateVisual"/> method.
00140 /// </summary>
00141         public SKMatrix?  Transform { get; set; } = null;
00142
00143 /// <summary>
00144 /// A tag to access the <see cref="SKRenderAction"/>.
00145 /// </summary>
00146         public string Tag { get; private set; }
00147
00148 /// <summary>
00149 /// The Z-index of the rendering action (an action with a higher Z-index will always appear above an
      action with a lower Z-index).
00150 /// The more different values there are for the Z-index, the slower the rendering, so keep use of this
      property to a minimum.
00151 /// If you change this, you probably want to call this object's <see cref="InvalidateZIndex"/> method.
00152 /// </summary>
00153         public uint ZIndex { get; set; } = 0;
00154
00155 /// <summary>
00156 /// The graphics that will be drawn with the specified <see cref="Filter"/>.  If you change this, you
      probably want to call this object's <see cref="InvalidateVisual"/> method.
00157 /// </summary>
00158         public SKRenderContext Graphics { get; set; } = null;
00159
00160 /// <summary>
00161 /// The filter with which the <see cref="Graphics"/> is drawn.  If you change this, you probably want
      to call this object's <see cref="InvalidateVisual"/> method.
00162 /// </summary>
00163         public IFilter Filter { get; set; }
00164
00165 /// <summary>
00166 /// An arbitrary object associated with the RenderAction.
00167 /// </summary>
00168         public object Payload { get; set; }
00169
00170         internal ISKRenderCanvas InternalParent { get; set; }
00171
00172 /// <summary>
00173 /// The container of this <see cref="SKRenderAction"/>.
00174 /// </summary>
00175         public Avalonia.Controls.Canvas Parent
00176         {
00177             get
00178             {
00179                 return (Avalonia.Controls.Canvas)InternalParent;
00180             }
00181         }
00182
00183 /// <summary>
00184 /// Raised when the pointer enters the area covered by the <see cref="SKRenderAction"/>.
00185 /// </summary>
00186         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerEntered;
00187
00188 /// <summary>
00189 /// Raised when the pointer leaves the area covered by the <see cref="SKRenderAction"/>.
00190 /// </summary>
00191         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerExited;
00192
00193 /// <summary>
00194 /// Raised when the pointer is pressed while over the area covered by the <see
      cref="SKRenderAction"/>.
00195 /// </summary>
00196         public event EventHandler<Avalonia.Input.PointerPressedEventArgs> PointerPressed;
00197
00198 /// <summary>
00199 /// Raised when the pointer is released after a <see cref="PointerPressed"/> event.
00200 /// </summary>
00201         public event EventHandler<Avalonia.Input.PointerReleasedEventArgs> PointerReleased;
00202
00203
00204         internal void FirePointerEntered(Avalonia.Input.PointerEventArgs e)
00205         {
00206             this.PointerEntered?.Invoke(this, e);
00207         }
00208
00209         internal void FirePointerExited(Avalonia.Input.PointerEventArgs e)
00210         {
00211             this.PointerExited?.Invoke(this, e);
00212         }
00213
00214         internal void FirePointerPressed(Avalonia.Input.PointerPressedEventArgs e)
00215         {
00216             this.PointerPressed?.Invoke(this, e);
00217         }
```

```
00218
00219         internal void FirePointerReleased(Avalonia.Input.PointerReleasedEventArgs e)
00220         {
00221             this.PointerReleased?.Invoke(this, e);
00222         }
00223
00224 /// <summary>
00225 /// <para>Signals to this object that its shape has changed and a new path needs to be computed for
      the purpose of hit-testing.
00226 /// Also signals to the <see cref="Parent"/> that the visual properties of this object have changed
      and triggers a redraw.</para>
00227 /// <para>This method should be called whenever the "shape" of the object represented by the <see
      cref="SKRenderAction"/> changes.
00228 /// If only the visual properties of this object have changed (e.g.  the colour), call the <see
      cref="InvalidateVisual"/> method instead.</para>
00229 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="SKMultiLayerRenderCanvas"/>, you only need to invalidate the last one.</para>
00230 /// <para>This method should only be called after the output has been fully initialized.</para>
00231 /// </summary>
00232         public void InvalidateHitTestPath()
00233         {
00234             CreateHitTestPath();
00235
00236             this.InvalidateVisual();
00237         }
00238
00239         internal void CreateHitTestPath()
00240         {
00241             if (this.ActionType == ActionTypes.Path)
00242             {
00243                 if (this.Path != null && this.Paint != null)
00244                 {
00245                     this.HitTestPath?.Dispose();
00246                     this.HitTestPath = this.Paint.GetFillPath(this.Path);
00247                 }
00248             }
00249             else if (this.ActionType == ActionTypes.RasterImage)
00250             {
00251                 if (this.ImageDestination != null)
00252                 {
00253                     this.HitTestPath?.Dispose();
00254                     this.HitTestPath = new SKPath();
00255                     this.HitTestPath.AddRect(this.ImageDestination.Value);
00256                 }
00257             }
00258             else if (this.ActionType == ActionTypes.Text)
00259             {
00260                 if (this.Paint != null && this.Font != null)
00261                 {
00262                     this.HitTestPath?.Dispose();
00263                     this.Paint.Typeface = this.Font.Typeface;
00264                     this.Paint.TextSize = this.Font.Size;
00265
00266                     using (SKPath pth = this.Paint.GetTextPath(this.Text, this.TextX, this.TextY))
00267                     {
00268                         this.HitTestPath = this.Paint.GetFillPath(pth);
00269                     }
00270                 }
00271             }
00272         }
00273
00274 /// <summary>
00275 /// <para>This methods signals to the <see cref="Parent"/> that the visual properties (e.g.  the
      colour) of this object have changed and triggers a redraw.</para>
00276 /// <para>If the "shape" of the object has changed as well, call the <see
      cref="InvalidateHitTestPath"/> method instead.  If the Z-index of the
00277 /// object has changed, call the <see cref="InvalidateZIndex"/> method instead.  If both the "shape"
      and the Z-index of the object have changed,
00278 /// call the <see cref="InvalidateAll"/> method.</para>
00279 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00280 /// <para>This method should only be called after the output has been fully initialized.</para>
00281 /// </summary>
00282         public void InvalidateVisual()
00283         {
00284             this.InternalParent?.InvalidateDirty();
00285         }
00286
00287 /// <summary>
00288 /// <para>This methods signals to the <see cref="Parent"/> that the Z-index and visual properties
      (e.g.  the colour) of this object have changed and triggers a redraw.</para>
00289 /// <para>If the "shape" of the object has changed as well, call the <see cref="InvalidateAll"/>
      method instead.</para>
00290 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00291 /// <para>This method should only be called after the output has been fully initialized.</para>
00292 /// </summary>
```

```
00293         public void InvalidateZIndex()
00294         {
00295             this.InternalParent?.InvalidateZIndex();
00296         }
00297 
00298 /// <summary>
00299 /// <para>This methods signals to the <see cref="Parent"/> that the Z-index, shape and visual
      properties (e.g.  the colour) of this object have changed and triggers a redraw.</para>
00300 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00301 /// <para>This method should only be called after the output has been fully initialized.</para>
00302 /// </summary>
00303         public void InvalidateAll()
00304         {
00305             this.InvalidateHitTestPath();
00306             this.InvalidateZIndex();
00307         }
00308 
00309         private SKRenderAction()
00310         {
00311 
00312         }
00313 
00314 /// <summary>
00315 /// Creates a new <see cref="SKRenderAction"/> representing a path.
00316 /// </summary>
00317 /// <param name="path">The geometry to be rendered.</param>
00318 /// <param name="paint">The paint used to fill or stroke the path.</param>
00319 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00320 /// <returns>A new <see cref="SKRenderAction"/> representing a path.</returns>
00321         public static SKRenderAction PathAction(SKPath path, SKPaint paint, string tag = null)
00322         {
00323             SKRenderAction act = new SKRenderAction()
00324             {
00325                 ActionType = ActionTypes.Path,
00326                 Path = path,
00327                 Paint = paint,
00328                 Tag = tag
00329             };
00330 
00331             if (!string.IsNullOrEmpty(tag))
00332             {
00333                 act.CreateHitTestPath();
00334             }
00335 
00336             return act;
00337         }
00338 
00339 /// <summary>
00340 /// Creates a new <see cref="SKRenderAction"/> representing a clipping action.
00341 /// </summary>
00342 /// <param name="clippingPath">The path to be used for clipping.</param>
00343 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00344 /// <returns>A new <see cref="SKRenderAction"/> representing a clipping action.</returns>
00345         public static SKRenderAction ClipAction(SKPath clippingPath, string tag = null)
00346         {
00347             return new SKRenderAction()
00348             {
00349                 ActionType = ActionTypes.Clip,
00350                 Path = clippingPath,
00351                 Tag = tag
00352             };
00353         }
00354 
00355 /// <summary>
00356 /// Creates a new <see cref="SKRenderAction"/> representing text.
00357 /// </summary>
00358 /// <param name="text">The text to be rendered.</param>
00359 /// <param name="x">The X coordinate at which the text will be drawn.</param>
00360 /// <param name="y">The Y coordinate at which the text will be drawn.</param>
00361 /// <param name="font">The font to be used to render the text.</param>
00362 /// <param name="paint">The paint to be used to fill or stroke the text.</param>
00363 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00364 /// <returns>A new <see cref="SKRenderAction"/> representing text.</returns>
00365         public static SKRenderAction TextAction(string text, float x, float y, SKFont font, SKPaint
      paint, string tag = null)
00366         {
00367             SKRenderAction act = new SKRenderAction()
00368             {
00369                 ActionType = ActionTypes.Text,
00370                 Text = text,
00371                 TextX = x,
00372                 TextY = y,
00373                 Font = font,
00374                 Paint = paint,
```

```
00375                 Tag = tag
00376             };
00377
00378             act.Paint.Typeface = font.Typeface;
00379             act.Paint.TextSize = font.Size;
00380
00381             if (!string.IsNullOrEmpty(tag))
00382             {
00383                 act.CreateHitTestPath();
00384             }
00385
00386             return act;
00387         }
00388
00389 /// <summary>
00390 /// Creates a new <see cref="SKRenderAction"/> representing an image.
00391 /// </summary>
00392 /// <param name="imageId">The univocal identifier of the image to draw.</param>
00393 /// <param name="sourceRect">The source rectangle of the image.</param>
00394 /// <param name="destinationRect">The destination rectangle of the image.</param>
00395 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00396 /// <returns>A new <see cref="SKRenderAction"/> representing an image.</returns>
00397         public static SKRenderAction ImageAction(string imageId, SKRect sourceRect, SKRect
      destinationRect, string tag = null)
00398         {
00399             SKRenderAction act = new SKRenderAction()
00400             {
00401                 ActionType = ActionTypes.RasterImage,
00402                 ImageId = imageId,
00403                 ImageSource = sourceRect,
00404                 ImageDestination = destinationRect,
00405                 Tag = tag
00406             };
00407
00408             if (!string.IsNullOrEmpty(tag))
00409             {
00410                 act.CreateHitTestPath();
00411             }
00412
00413             return act;
00414         }
00415
00416 /// <summary>
00417 /// Creates a new <see cref="SKRenderAction"/> representing a transform.
00418 /// </summary>
00419 /// <param name="transform">The transform to apply.</param>
00420 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00421 /// <returns>A new <see cref="SKRenderAction"/> representing a transform.</returns>
00422         public static SKRenderAction TransformAction(SKMatrix transform, string tag = null)
00423         {
00424             return new SKRenderAction()
00425             {
00426                 ActionType = ActionTypes.Transform,
00427                 Transform = transform,
00428                 Tag = tag
00429             };
00430         }
00431
00432 /// <summary>
00433 /// Creates a new <see cref="SKRenderAction"/> that saves the current graphics state.
00434 /// </summary>
00435 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00436 /// <returns>A new <see cref="SKRenderAction"/> that saves the current graphics state.</returns>
00437         public static SKRenderAction SaveAction(string tag = null)
00438         {
00439             return new SKRenderAction()
00440             {
00441                 ActionType = ActionTypes.Save,
00442                 Tag = tag
00443             };
00444         }
00445
00446 /// <summary>
00447 /// Creates a new <see cref="SKRenderAction"/> that saves the current graphics state.
00448 /// </summary>
00449 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00450 /// <returns>A new <see cref="SKRenderAction"/> that restores the last saved graphics state.</returns>
00451         public static SKRenderAction RestoreAction(string tag = null)
00452         {
00453             return new SKRenderAction()
00454             {
00455                 ActionType = ActionTypes.Restore,
00456                 Tag = tag
00457             };
00458         }
00459
```

```
00460 /// <summary>
00461 /// Create a new <see cref="SKRenderAction"/> that draws some graphics with a filter.
00462 /// </summary>
00463 /// <returns>A new <see cref="SKRenderAction"/> that draws some graphics with a filter.</returns>
00464         public static SKRenderAction DrawFilteredGraphicsAction(SKRenderContext graphics, IFilter
      filter, string tag = null)
00465         {
00466             return new SKRenderAction()
00467             {
00468                 ActionType = ActionTypes.DrawFiltered,
00469                 Graphics = graphics,
00470                 Filter = filter,
00471                 Tag = tag
00472             };
00473         }
00474 
00475 /// <inheritdoc cref="IDisposable.Dispose"/>
00476         protected virtual void Dispose(bool disposing)
00477         {
00478             if (!disposedValue)
00479             {
00480                 if (disposing)
00481                 {
00482                     this.Font?.Dispose();
00483                     this.HitTestPath?.Dispose();
00484                     this.LastRenderedGlobalHitTestPath?.Dispose();
00485                     this.Paint?.Dispose();
00486                     this.Path?.Dispose();
00487                 }
00488 
00489                 disposedValue = true;
00490             }
00491         }
00492 
00493 /// <inheritdoc cref="IDisposable.Dispose"/>
00494         public void Dispose()
00495         {
00496             Dispose(disposing:  true);
00497             GC.SuppressFinalize(this);
00498         }
00499     }
00500 
00501 
00502     internal static class SKTypefaceCache
00503     {
00504         private static readonly object LockObject = new object();
00505         private static readonly Dictionary<string, SKTypeface> Typefaces = new Dictionary<string,
      SKTypeface>();
00506         public static SKTypeface GetSKTypeface(FontFamily family)
00507         {
00508 
00509             lock (LockObject)
00510             {
00511                 if (Typefaces.TryGetValue(family.FileName, out SKTypeface tbr))
00512                 {
00513                     return tbr;
00514                 }
00515                 else
00516                 {
00517                     try
00518                     {
00519                         System.IO.MemoryStream fontStream = new
      System.IO.MemoryStream((int)family.TrueTypeFile.FontStream.Length);
00520                         family.TrueTypeFile.FontStream.Seek(0, System.IO.SeekOrigin.Begin);
00521                         family.TrueTypeFile.FontStream.CopyTo(fontStream);
00522                         fontStream.Seek(0, System.IO.SeekOrigin.Begin);
00523 
00524                         SKTypeface typeface = SKTypeface.FromData(SKData.Create(fontStream));
00525 
00526                         Typefaces[family.FileName] = typeface;
00527 
00528                         return typeface;
00529                     }
00530                     catch
00531                     {
00532                         SKTypeface typeface = SKTypeface.Default;
00533 
00534                         Typefaces[family.FileName] = typeface;
00535 
00536                         return typeface;
00537                     }
00538                 }
00539             }
00540         }
00541     }
00542 
00543 /// <summary>
```

```
00544 /// Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.
00545 /// </summary>
00546     public class SKRenderContext
00547     {
00548         internal virtual Dictionary<string, (SKBitmap, bool)> Images { get; set; }
00549         internal virtual List<SKRenderAction> SKRenderActions { get; set; }
00550     }
00551 
00552 
00553     internal class SKRenderContextImpl :  SKRenderContext, IGraphicsContext, IDisposable
00554     {
00555         public Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> TaggedActions {
      get; set; } = new Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>>();
00556 
00557         private readonly bool removeTaggedActions = true;
00558 
00559         public string Tag { get; set; }
00560 
00561         private readonly AvaloniaContextInterpreter.TextOptions _textOption;
00562 
00563         internal override Dictionary<string, (SKBitmap, bool)> Images { get; set; }
00564 
00565         private readonly FilterOption _filterOption;
00566 
00567         public SKRenderContextImpl(double width, double height, bool
      removeTaggedActionsAfterExecution, AvaloniaContextInterpreter.TextOptions textOption,
      Dictionary<string, (SKBitmap, bool)> images, FilterOption filterOption)
00568         {
00569             this.Images = images;
00570 
00571             currentPath = null;
00572             figureInitialised = false;
00573 
00574             SKRenderActions = new List<SKRenderAction>();
00575             removeTaggedActions = removeTaggedActionsAfterExecution;
00576 
00577             Width = width;
00578             Height = height;
00579 
00580             _textOption = textOption;
00581             _filterOption = filterOption;
00582         }
00583 
00584         internal override List<SKRenderAction> SKRenderActions { get; set; }
00585 
00586         public double Width { get; private set; }
00587         public double Height { get; private set; }
00588 
00589         private void AddAction(SKRenderAction act)
00590         {
00591             if (!string.IsNullOrEmpty(Tag))
00592             {
00593                 if (TaggedActions.ContainsKey(Tag))
00594                 {
00595                     IEnumerable<SKRenderAction> actions = TaggedActions[Tag](act);
00596 
00597                     foreach (SKRenderAction action in actions)
00598                     {
00599                         SKRenderActions.Add(action);
00600                     }
00601 
00602                     if (removeTaggedActions)
00603                     {
00604                         TaggedActions.Remove(Tag);
00605                     }
00606                 }
00607                 else
00608                 {
00609                     SKRenderActions.Add(act);
00610                 }
00611             }
00612             else if (TaggedActions.ContainsKey(""))
00613             {
00614                 IEnumerable<SKRenderAction> actions = TaggedActions[""](act);
00615 
00616                 foreach (SKRenderAction action in actions)
00617                 {
00618                     SKRenderActions.Add(action);
00619                 }
00620             }
00621             else
00622             {
00623                 SKRenderActions.Add(act);
00624             }
00625         }
00626 
00627         public void Translate(double x, double y)
```

```
00628         {
00629             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00630 
00631             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateTranslation((float)x,
      (float)y), Tag);
00632             AddAction(act);
00633 
00634             currentPath = null;
00635             figureInitialised = false;
00636         }
00637 
00638         public TextBaselines TextBaseline { get; set; }
00639 
00640         private void PathText(string text, double x, double y)
00641         {
00642             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00643 
00644             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00645 
00646             for (int j = 0; j < textPath.Segments.Count; j++)
00647             {
00648                 switch (textPath.Segments[j].Type)
00649                 {
00650                     case VectSharp.SegmentType.Move:
00651                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00652                         break;
00653                     case VectSharp.SegmentType.Line:
00654                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00655                         break;
00656                     case VectSharp.SegmentType.CubicBezier:
00657                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00658                         break;
00659                     case VectSharp.SegmentType.Close:
00660                         this.Close();
00661                         break;
00662                 }
00663             }
00664         }
00665 
00666         public void StrokeSimpleText(string text, double x, double y)
00667         {
00668             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00669 
00670             if ((_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && (Font.FontFamily.IsStandardFamily ||
      Font.FontFamily.TrueTypeFile?.FontStream != null))) &&
      !System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(System.Runtime.InteropServices.OSPlatform.Linux))
00671             {
00672                 SKTypeface typeface = SKTypefaceCache.GetSKTypeface(Font.FontFamily);
00673 
00674                 double top = y;
00675                 double left = x;
00676 
00677                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00678 
00679                 if (TextBaseline == TextBaselines.Top)
00680                 {
00681                     if (Font.FontFamily.TrueTypeFile != null)
00682                     {
00683                         left -= metrics.LeftSideBearing;
00684                         top += metrics.Top;
00685                     }
00686                 }
00687                 else if (TextBaseline == TextBaselines.Middle)
00688                 {
00689                     if (Font.FontFamily.TrueTypeFile != null)
00690                     {
00691                         left -= metrics.LeftSideBearing;
00692                         top += (metrics.Top + metrics.Bottom) * 0.5;
00693                     }
00694                 }
00695                 else if (TextBaseline == TextBaselines.Baseline)
00696                 {
00697                     if (Font.FontFamily.TrueTypeFile != null)
00698                     {
00699                         left -= metrics.LeftSideBearing;
00700                     }
00701                 }
00702                 else if (TextBaseline == TextBaselines.Bottom)
00703                 {
00704                     if (Font.FontFamily.TrueTypeFile != null)
00705                     {
00706                         left -= metrics.LeftSideBearing;
00707                         top += metrics.Bottom;
00708                     }
```

```
00709                 }
00710
00711                 SKPaint stroke = new SKPaint() { IsStroke = true, IsAntialias = true, Style =
      SKPaintStyle.Stroke, StrokeWidth = (float)LineWidth, SubpixelText = true };
00712
00713                 if (this.StrokeStyle is SolidColourBrush solid)
00714                 {
00715                     stroke.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), StrokeAlpha);
00716                 }
00717                 else if (this.StrokeStyle is LinearGradientBrush linearGradient)
00718                 {
00719                     stroke.Shader = linearGradient.ToSKShader();
00720                 }
00721                 else if (this.StrokeStyle is RadialGradientBrush radialGradient)
00722                 {
00723                     stroke.Shader = radialGradient.ToSKShader();
00724                 }
00725
00726                 stroke.PathEffect = SKPathEffect.CreateDash(new float[] { (float)LineDash[0],
      (float)LineDash[1] }, (float)LineDash[2]);
00727
00728                 switch (LineCap)
00729                 {
00730                     case LineCaps.Butt:
00731                         stroke.StrokeCap = SKStrokeCap.Butt;
00732                         break;
00733                     case LineCaps.Round:
00734                         stroke.StrokeCap = SKStrokeCap.Round;
00735                         break;
00736                     case LineCaps.Square:
00737                         stroke.StrokeCap = SKStrokeCap.Square;
00738                         break;
00739                 }
00740
00741                 switch (LineJoin)
00742                 {
00743                     case LineJoins.Bevel:
00744                         stroke.StrokeJoin = SKStrokeJoin.Bevel;
00745                         break;
00746                     case LineJoins.Round:
00747                         stroke.StrokeJoin = SKStrokeJoin.Round;
00748                         break;
00749                     case LineJoins.Miter:
00750                         stroke.StrokeJoin = SKStrokeJoin.Miter;
00751                         break;
00752                 }
00753
00754                 SKRenderAction act = SKRenderAction.TextAction(text, (float)left, (float)top, new
      SKFont(typeface, (float)Font.FontSize), stroke, Tag);
00755
00756                 AddAction(act);
00757             }
00758             else
00759             {
00760                 PathText(text, x, y);
00761                 Stroke();
00762             }
00763         }
00764
00765         public void FillText(string text, double x, double y)
00766         {
00767             if (!Font.EnableKerning)
00768             {
00769                 FillSimpleText(text, x, y);
00770             }
00771             else
00772             {
00773                 List<(string, Point)> tSpans = new List<(string, Point)>();
00774
00775                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
00776                 Point currentKerning = new Point();
00777
00778                 Point currentGlyphPlacementDelta = new Point();
00779                 Point currentGlyphAdvanceDelta = new Point();
00780                 Point nextGlyphPlacementDelta = new Point();
00781                 Point nextGlyphAdvanceDelta = new Point();
00782
00783                 for (int i = 0; i < text.Length; i++)
00784                 {
00785                     if (i < text.Length - 1)
00786                     {
00787                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00788                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00789                         nextGlyphAdvanceDelta = new Point();
00790                         nextGlyphPlacementDelta = new Point();
00791
```

```
00792                       TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00793
00794                       if (kerning != null)
00795                       {
00796                           currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00797                           currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00798
00799                           nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00800                           nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00801                       }
00802                   }
00803
00804                   if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00805                   {
00806                       if (currentRun.Length > 0)
00807                       {
00808                           tSpans.Add((currentRun.ToString(), currentKerning));
00809
00810                           tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
00811
00812                           currentRun.Clear();
00813                           currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00814                       }
00815                       else
00816                       {
00817                           tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      currentKerning.Y)));
00818
00819                           currentRun.Clear();
00820                           currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00821                       }
00822                   }
00823                   else
00824                   {
00825                       currentRun.Append(text[i]);
00826                   }
00827               }
00828
00829               if (currentRun.Length > 0)
00830               {
00831                   tSpans.Add((currentRun.ToString(), currentKerning));
00832               }
00833
00834               double currX = x;
00835               double currY = y;
00836
00837               Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
00838
00839               if (TextBaseline == TextBaselines.Top)
00840               {
00841                   if (Font.FontFamily.TrueTypeFile != null)
00842                   {
00843                       currY += fullMetrics.Top;
00844                   }
00845               }
00846               else if (TextBaseline == TextBaselines.Middle)
00847               {
00848                   if (Font.FontFamily.TrueTypeFile != null)
00849                   {
00850                       currY += (fullMetrics.Top + fullMetrics.Bottom) * 0.5;
00851                   }
00852               }
00853               else if (TextBaseline == TextBaselines.Bottom)
00854               {
00855                   if (Font.FontFamily.TrueTypeFile != null)
00856                   {
00857                       currY += fullMetrics.Bottom;
00858                   }
00859               }
00860
00861               TextBaseline = TextBaselines.Baseline;
00862
00863               for (int i = 0; i < tSpans.Count; i++)
00864               {
00865                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
```

```
00866
00867                     if (i == 0)
00868                     {
00869                         FillSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
    tSpans[i].Item2.Y);
00870                     }
00871                     else
00872                     {
00873                         FillSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
    fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
00874                     }
00875
00876
00877                     currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
00878                     currY += tSpans[i].Item2.Y;
00879                 }
00880             }
00881         }
00882
00883         public void StrokeText(string text, double x, double y)
00884         {
00885             if (!Font.EnableKerning)
00886             {
00887                 StrokeSimpleText(text, x, y);
00888             }
00889             else
00890             {
00891                 List<(string, Point)> tSpans = new List<(string, Point)>();
00892
00893                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
00894                 Point currentKerning = new Point();
00895
00896                 Point currentGlyphPlacementDelta = new Point();
00897                 Point currentGlyphAdvanceDelta = new Point();
00898                 Point nextGlyphPlacementDelta = new Point();
00899                 Point nextGlyphAdvanceDelta = new Point();
00900
00901                 for (int i = 0; i < text.Length; i++)
00902                 {
00903                     if (i < text.Length - 1)
00904                     {
00905                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00906                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00907                         nextGlyphAdvanceDelta = new Point();
00908                         nextGlyphPlacementDelta = new Point();
00909
00910                         TrueTypeFile.PairKerning kerning =
    Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00911
00912                         if (kerning != null)
00913                         {
00914                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
    kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00915                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
    kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00916
00917                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
    kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00918                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00919                         }
00920                     }
00921
00922                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
    currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00923                     {
00924                         if (currentRun.Length > 0)
00925                         {
00926                             tSpans.Add((currentRun.ToString(), currentKerning));
00927
00928                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
    Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
00929
00930                             currentRun.Clear();
00931                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
    currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00932                         }
00933                         else
00934                         {
00935                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
    Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
    currentKerning.Y)));
00936
00937                             currentRun.Clear();
00938                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
```

```
    currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00939                       }
00940                   }
00941                   else
00942                   {
00943                       currentRun.Append(text[i]);
00944                   }
00945               }
00946
00947               if (currentRun.Length > 0)
00948               {
00949                   tSpans.Add((currentRun.ToString(), currentKerning));
00950               }
00951
00952               double currX = x;
00953               double currY = y;
00954
00955               Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
00956
00957               for (int i = 0; i < tSpans.Count; i++)
00958               {
00959                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
00960
00961                   if (i == 0)
00962                   {
00963                       StrokeSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
    tSpans[i].Item2.Y);
00964                   }
00965                   else
00966                   {
00967                       StrokeSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
    fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
00968                   }
00969
00970
00971                   currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
00972                   currY += tSpans[i].Item2.Y;
00973               }
00974           }
00975       }
00976
00977
00978       public void FillSimpleText(string text, double x, double y)
00979       {
00980           Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00981
00982           if ((_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
    AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && (Font.FontFamily.IsStandardFamily ||
    Font.FontFamily.TrueTypeFile?.FontStream != null))) &&
    !System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(System.Runtime.InteropServices.OSPlatform.Linux))
00983           {
00984               SKTypeface typeface = SKTypefaceCache.GetSKTypeface(Font.FontFamily);
00985
00986               double top = y;
00987               double left = x;
00988
00989               Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00990
00991               if (TextBaseline == TextBaselines.Top)
00992               {
00993                   if (Font.FontFamily.TrueTypeFile != null)
00994                   {
00995                       left -= metrics.LeftSideBearing;
00996                       top += metrics.Top;
00997                   }
00998               }
00999               else if (TextBaseline == TextBaselines.Middle)
01000               {
01001                   if (Font.FontFamily.TrueTypeFile != null)
01002                   {
01003                       left -= metrics.LeftSideBearing;
01004                       top += (metrics.Top + metrics.Bottom) * 0.5;
01005                   }
01006               }
01007               else if (TextBaseline == TextBaselines.Baseline)
01008               {
01009                   if (Font.FontFamily.TrueTypeFile != null)
01010                   {
01011                       left -= metrics.LeftSideBearing;
01012                   }
01013               }
01014               else if (TextBaseline == TextBaselines.Bottom)
01015               {
01016                   if (Font.FontFamily.TrueTypeFile != null)
01017                   {
01018                       left -= metrics.LeftSideBearing;
01019                       top += metrics.Bottom;
```

```
01020                     }
01021                 }
01022
01023                 SKPaint fill = new SKPaint() { IsStroke = false, IsAntialias = true, Style =
      SKPaintStyle.Fill, SubpixelText = true };
01024
01025                 if (this.FillStyle is SolidColourBrush solid)
01026                 {
01027                     fill.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), FillAlpha);
01028                 }
01029                 else if (this.FillStyle is LinearGradientBrush linearGradient)
01030                 {
01031                     fill.Shader = linearGradient.ToSKShader();
01032                 }
01033                 else if (this.FillStyle is RadialGradientBrush radialGradient)
01034                 {
01035                     fill.Shader = radialGradient.ToSKShader();
01036                 }
01037
01038                 SKRenderAction act = SKRenderAction.TextAction(text, (float)left, (float)top, new
      SKFont(typeface, (float)Font.FontSize), fill, Tag);
01039
01040                 AddAction(act);
01041             }
01042             else
01043             {
01044                 PathText(text, x, y);
01045                 Fill(FillRule.NonZeroWinding);
01046             }
01047         }
01048
01049         public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
01050         private byte StrokeAlpha = 255;
01051
01052         public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
01053         private byte FillAlpha = 255;
01054
01055         public void SetFillStyle((int r, int g, int b, double a) style)
01056         {
01057             FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
01058             FillAlpha = (byte)(style.a * 255);
01059         }
01060
01061         public void SetFillStyle(Brush style)
01062         {
01063             FillStyle = style;
01064
01065             if (style is SolidColourBrush solid)
01066             {
01067                 FillAlpha = (byte)(solid.A * 255);
01068             }
01069             else
01070             {
01071                 FillAlpha = 255;
01072             }
01073         }
01074
01075         public void SetStrokeStyle((int r, int g, int b, double a) style)
01076         {
01077             StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
01078             StrokeAlpha = (byte)(style.a * 255);
01079         }
01080
01081         public void SetStrokeStyle(Brush style)
01082         {
01083             StrokeStyle = style;
01084
01085             if (style is SolidColourBrush solid)
01086             {
01087                 StrokeAlpha = (byte)(solid.A * 255);
01088             }
01089             else
01090             {
01091                 StrokeAlpha = 255;
01092             }
01093         }
01094
01095         private double[] LineDash;
01096
01097         public void SetLineDash(LineDash dash)
01098         {
01099             LineDash = new double[] { dash.UnitsOn, dash.UnitsOff, dash.Phase };
01100         }
01101
01102         public void Rotate(double angle)
01103         {
```

```
01104             Utils.CoerceNaNAndInfinityToZero(ref angle);
01105
01106             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateRotation((float)angle),
      Tag);
01107             AddAction(act);
01108
01109             currentPath = null;
01110             figureInitialised = false;
01111         }
01112
01113         public void Transform(double a, double b, double c, double d, double e, double f)
01114         {
01115             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
01116
01117             SKRenderAction act = SKRenderAction.TransformAction(new SKMatrix((float)a, (float)c,
      (float)e, (float)b, (float)d, (float)f, 0, 0, 1), Tag);
01118             AddAction(act);
01119
01120             currentPath = null;
01121             figureInitialised = false;
01122         }
01123
01124         public void Scale(double x, double y)
01125         {
01126             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01127
01128             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateScale((float)x,
      (float)y), Tag);
01129             AddAction(act);
01130
01131             currentPath = null;
01132             figureInitialised = false;
01133         }
01134
01135         public void Save()
01136         {
01137             SKRenderAction act = SKRenderAction.SaveAction(Tag);
01138             AddAction(act);
01139
01140             currentPath = null;
01141             figureInitialised = false;
01142         }
01143
01144         public void Restore()
01145         {
01146             SKRenderAction act = SKRenderAction.RestoreAction(Tag);
01147             AddAction(act);
01148
01149             currentPath = null;
01150             figureInitialised = false;
01151         }
01152
01153         public double LineWidth { get; set; }
01154         public LineCaps LineCap { get; set; }
01155         public LineJoins LineJoin { get; set; }
01156
01157         public Font Font { get; set; }
01158
01159         private SKPath currentPath;
01160
01161         private bool figureInitialised = false;
01162         private bool disposedValue;
01163
01164         public void MoveTo(double x, double y)
01165         {
01166             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01167
01168             if (currentPath == null)
01169             {
01170                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01171             }
01172
01173             currentPath.MoveTo((float)x, (float)y);
01174             figureInitialised = true;
01175         }
01176
01177         public void LineTo(double x, double y)
01178         {
01179             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01180
01181             if (currentPath == null)
01182             {
01183                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01184             }
01185
01186             if (!figureInitialised)
01187             {
```

```
01188                 figureInitialised = true;
01189                 currentPath.MoveTo((float)x, (float)y);
01190             }
01191             else
01192             {
01193                 currentPath.LineTo((float)x, (float)y);
01194             }
01195         }
01196 
01197         public void Rectangle(double x0, double y0, double width, double height)
01198         {
01199             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
01200 
01201             if (currentPath == null)
01202             {
01203                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01204             }
01205 
01206             currentPath.MoveTo((float)x0, (float)y0);
01207             currentPath.LineTo((float)(x0 + width), (float)y0);
01208             currentPath.LineTo((float)(x0 + width), (float)(y0 + height));
01209             currentPath.LineTo((float)x0, (float)(y0 + height));
01210 
01211             currentPath.Close();
01212             figureInitialised = false;
01213         }
01214 
01215         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
01216         {
01217             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
01218 
01219             if (currentPath == null)
01220             {
01221                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01222             }
01223 
01224             if (figureInitialised)
01225             {
01226                 currentPath.CubicTo((float)p1X, (float)p1Y, (float)p2X, (float)p2Y, (float)p3X,
      (float)p3Y);
01227             }
01228             else
01229             {
01230                 currentPath.MoveTo((float)p1X, (float)p1Y);
01231                 figureInitialised = true;
01232             }
01233         }
01234 
01235         public void Close()
01236         {
01237             currentPath.Close();
01238 
01239             figureInitialised = false;
01240         }
01241 
01242         public void Stroke()
01243         {
01244             SKPaint stroke = new SKPaint() { IsStroke = true, IsAntialias = true, Style =
      SKPaintStyle.Stroke, StrokeWidth = (float)LineWidth };
01245 
01246             if (this.StrokeStyle is SolidColourBrush solid)
01247             {
01248                 stroke.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), StrokeAlpha);
01249             }
01250             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
01251             {
01252                 stroke.Shader = linearGradient.ToSKShader();
01253             }
01254             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
01255             {
01256                 stroke.Shader = radialGradient.ToSKShader();
01257             }
01258 
01259             stroke.PathEffect = SKPathEffect.CreateDash(new float[] { (float)LineDash[0],
      (float)LineDash[1] }, (float)LineDash[2]);
01260 
01261             switch (LineCap)
01262             {
01263                 case LineCaps.Butt:
01264                     stroke.StrokeCap = SKStrokeCap.Butt;
01265                     break;
01266                 case LineCaps.Round:
01267                     stroke.StrokeCap = SKStrokeCap.Round;
01268                     break;
01269                 case LineCaps.Square:
```

```
01270                       stroke.StrokeCap = SKStrokeCap.Square;
01271                       break;
01272               }
01273
01274               switch (LineJoin)
01275               {
01276                   case LineJoins.Bevel:
01277                       stroke.StrokeJoin = SKStrokeJoin.Bevel;
01278                       break;
01279                   case LineJoins.Round:
01280                       stroke.StrokeJoin = SKStrokeJoin.Round;
01281                       break;
01282                   case LineJoins.Miter:
01283                       stroke.StrokeJoin = SKStrokeJoin.Miter;
01284                       break;
01285               }
01286
01287               SKRenderAction act = SKRenderAction.PathAction(currentPath, stroke, Tag);
01288
01289               AddAction(act);
01290
01291               currentPath = null;
01292               figureInitialised = false;
01293           }
01294
01295           public void Fill(FillRule fillRule)
01296           {
01297               SKPaint fill = new SKPaint() { IsStroke = false, IsAntialias = true, Style =
      SKPaintStyle.Fill };
01298
01299               if (this.FillStyle is SolidColourBrush solid)
01300               {
01301                   fill.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255), (byte)(solid.B
      * 255), FillAlpha);
01302               }
01303               else if (this.FillStyle is LinearGradientBrush linearGradient)
01304               {
01305                   fill.Shader = linearGradient.ToSKShader();
01306               }
01307               else if (this.FillStyle is RadialGradientBrush radialGradient)
01308               {
01309                   fill.Shader = radialGradient.ToSKShader();
01310               }
01311
01312               switch (fillRule)
01313               {
01314                   case FillRule.NonZeroWinding:
01315                       currentPath.FillType = SKPathFillType.Winding;
01316                       break;
01317
01318                   case FillRule.EvenOdd:
01319                       currentPath.FillType = SKPathFillType.EvenOdd;
01320                       break;
01321               }
01322
01323               SKRenderAction act = SKRenderAction.PathAction(currentPath, fill, Tag);
01324
01325               AddAction(act);
01326
01327               currentPath = null;
01328               figureInitialised = false;
01329           }
01330
01331           public void SetClippingPath()
01332           {
01333               SKRenderAction act = SKRenderAction.ClipAction(currentPath, Tag);
01334
01335               AddAction(act);
01336
01337               currentPath = null;
01338               figureInitialised = false;
01339           }
01340
01341           public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
01342           {
01343               Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
01344
01345               if (!this.Images.ContainsKey(image.Id))
01346               {
01347                   SKBitmap bmp = SKBitmap.Decode(image.PNGStream);
01348                   this.Images.Add(image.Id, (bmp, image.Interpolate));
01349               }
01350
01351               SKRenderAction act = SKRenderAction.ImageAction(image.Id, new SKRect(sourceX, sourceY,
```

```
      sourceX + sourceWidth, sourceY + sourceHeight), new SKRect((float)destinationX, (float)destinationY,
      (float)(destinationX + destinationWidth), (float)(destinationY + destinationHeight)), Tag);
01352
01353             AddAction(act);
01354         }
01355
01356         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
01357         {
01358             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
01359             {
01360                 double scale = this._filterOption.RasterisationResolution;
01361
01362                 Rectangle bounds = graphics.GetBounds();
01363
01364                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
01365
01366                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
01367                 {
01368                     if (!this._filterOption.RasterisationResolutionRelative)
01369                     {
01370                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
01371                     }
01372
01373                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
01374                     RasterImage filtered = null;
01375
01376                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
01377                     {
01378                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
01379                     }
01380
01381                     if (filter is ILocationInvariantFilter locInvFilter)
01382                     {
01383                         filtered = locInvFilter.Filter(rasterised, scale);
01384                     }
01385                     else if (filter is IFilterWithLocation filterWithLoc)
01386                     {
01387                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
01388                     }
01389
01390                     if (filtered != null)
01391                     {
01392                         rasterised.Dispose();
01393
01394                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
01395                     }
01396                 }
01397             }
01398             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
01399             {
01400                 double scale = this._filterOption.RasterisationResolution;
01401
01402                 Rectangle bounds = graphics.GetBounds();
01403
01404                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
01405
01406                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
01407                 {
01408                     if (!this._filterOption.RasterisationResolutionRelative)
01409                     {
01410                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
01411                     }
01412
01413                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
01414                     {
01415                         RasterImage filtered = null;
01416
01417                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
01418                         {
01419
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
01420                         }
01421
01422                         if (filter is ILocationInvariantFilter locInvFilter)
01423                         {
01424                             filtered = locInvFilter.Filter(rasterised, scale);
01425                         }
01426                         else if (filter is IFilterWithLocation filterWithLoc)
01427                         {
```

```
01428                               filtered = filterWithLoc.Filter(rasterised, bounds, scale);
01429                           }
01430
01431                           if (filtered != null)
01432                           {
01433                               rasterised.Dispose();
01434
01435                               DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
01436                           }
01437                       }
01438                       else
01439                       {
01440                           throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
01441 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
01442 • Provide your own implementation of Graphics.RasterisationMethod.
01443 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
01444                       }
01445                   }
01446               }
01447               else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
01448               {
01449                   graphics.CopyToIGraphicsContext(this);
01450               }
01451               else
01452               {
01453
01454               }
01455           }
01456
01457           protected virtual void Dispose(bool disposing)
01458           {
01459               if (!disposedValue)
01460               {
01461                   if (disposing)
01462                   {
01463                       this.currentPath?.Dispose();
01464                   }
01465
01466                   disposedValue = true;
01467               }
01468           }
01469
01470           public void Dispose()
01471           {
01472               Dispose(disposing:  true);
01473               GC.SuppressFinalize(this);
01474           }
01475       }
01476
01477 /// <summary>
01478 /// Determines how and whether image filters are rasterised.
01479 /// </summary>
01480       public class FilterOption
01481       {
01482 /// <summary>
01483 /// Defines whether image filters should be rasterised or not.
01484 /// </summary>
01485           public enum FilterOperations
01486           {
01487 /// <summary>
01488 /// Image filters will always be rasterised using the SkiaSharp backend.
01489 /// </summary>
01490               RasteriseAllWithSkia,
01491
01492 /// <summary>
01493 /// Image filters will always be rasterised using the VectSharp.Raster or VectSharp.Raster.ImageSharp.
      This option requires a reference to VectSharp.Raster or to VectSharp.Raster.ImageSharp to be added.
01494 /// </summary>
01495               RasteriseAllWithVectSharp,
01496
01497 /// <summary>
01498 /// All image filters will be ignored.
01499 /// </summary>
01500               IgnoreAll,
01501
01502 /// <summary>
01503 /// All the images that should be drawn with a filter will be ignored.
01504 /// </summary>
01505               SkipAll
01506           }
01507
01508 /// <summary>
01509 /// Defines whether image filters should be rasterised or not.
01510 /// </summary>
```

```
01511         public FilterOperations Operation { get; } = FilterOperations.RasteriseAllWithSkia;
01512
01513 /// <summary>
01514 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
01515 /// </summary>
01516         public double RasterisationResolution { get; } = 1;
01517
01518 /// <summary>
01519 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
01520 /// </summary>
01521         public bool RasterisationResolutionRelative { get; } = true;
01522
01523 /// <summary>
01524 /// The default options for image filter rasterisation.
01525 /// </summary>
01526         public static FilterOption Default = new FilterOption(FilterOperations.RasteriseAllWithSkia,
      1, true);
01527
01528 /// <summary>
01529 /// Create a new <see cref="FilterOption"/> object.
01530 /// </summary>
01531 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
01532 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
01533 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
01534         public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
01535         {
01536             this.Operation = operation;
01537             this.RasterisationResolution = rasterisationResolution;
01538             this.RasterisationResolutionRelative = rasterisationResolutionRelative;
01539         }
01540     }
01541
01542 /// <summary>
01543 /// Contains methods to render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using
      the SkiaSharp renderer.
01544 /// </summary>
01545     public static class SKRenderContextInterpreter
01546     {
01547         internal static SKColor ToSKColor(this Colour colour)
01548         {
01549             return new SKColor((byte)(255 * colour.R), (byte)(255 * colour.G), (byte)(255 * colour.B),
      (byte)(255 * colour.A));
01550         }
01551
01552         internal static SKShader ToSKShader(this LinearGradientBrush brush)
01553         {
01554             return SKShader.CreateLinearGradient(new SKPoint((float)brush.StartPoint.X,
      (float)brush.StartPoint.Y), new SKPoint((float)brush.EndPoint.X, (float)brush.EndPoint.Y), (from el in
      brush.GradientStops select el.Colour.ToSKColor()).ToArray(), (from el in brush.GradientStops select
      (float)el.Offset).ToArray(), SKShaderTileMode.Clamp);
01555         }
01556
01557         internal static SKShader ToSKShader(this RadialGradientBrush brush)
01558         {
01559             return SKShader.CreateTwoPointConicalGradient(new SKPoint((float)brush.FocalPoint.X,
      (float)brush.FocalPoint.Y), 0, new SKPoint((float)brush.Centre.X, (float)brush.Centre.Y),
      (float)brush.Radius, (from el in brush.GradientStops select el.Colour.ToSKColor()).ToArray(), (from el
      in brush.GradientStops select (float)el.Offset).ToArray(), SKShaderTileMode.Clamp);
01560         }
01561
01562 /// <summary>
01563 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
      renderer.  Each page corresponds to a layer in the image.
01564 /// </summary>
01565 /// <param name="document">The <see cref="Document"/> to render.</param>
01566 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
      the largest page is used.</param>
01567 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
      of the largest page is used.</param>
01568 /// <param name="background">The background colour of the document.  If this is <see langword="null"
      />, a transparent background is used.</param>
01569 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01570 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01571 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01572         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document, double?  width
```

```
= null, double?  height = null, Colour?  background = null, AvaloniaContextInterpreter.TextOptions
textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption =
default)
01573         {
01574             filterOption = filterOption ??  FilterOption.Default;
01575 
01576             return new SKMultiLayerRenderCanvas((from el in document.Pages select
el.CopyToSKRenderContext(textOption, filterOption)).ToList(), (from el in document.Pages select
SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
select el.Height).Max());
01577         }
01578 
01579 /// <summary>
01580 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
renderer.  Each page corresponds to a layer in the image.
01581 /// </summary>
01582 /// <param name="document">The <see cref="Document"/> to render.</param>
01583 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01584 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01585 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
01586 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
the largest page is used.</param>
01587 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
of the largest page is used.</param>
01588 /// <param name="background">The background colour of the document.  If this is <see langword="null"
/>, a transparent background is used.</param>
01589 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01590 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01591 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01592         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document,
Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
removeTaggedActionsAfterExecution = true, double?  width = null, double?  height = null, Colour?
background = null, AvaloniaContextInterpreter.TextOptions textOption =
AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01593         {
01594             filterOption = filterOption ??  FilterOption.Default;
01595 
01596             return new SKMultiLayerRenderCanvas((from el in document.Pages select
el.CopyToSKRenderContext(taggedActions, removeTaggedActionsAfterExecution, textOption,
filterOption)).ToList(), (from el in document.Pages select
SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
select el.Height).Max());
01597         }
01598 
01599 /// <summary>
01600 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
renderer.  Each page corresponds to a layer in the image.
01601 /// </summary>
01602 /// <param name="document">The <see cref="Document"/> to render.</param>
01603 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01604 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01605 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
as "pixelated" or not.  This will be populated automatically as the page is rendered.
01606 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
times), it will be beneficial to keep a reference to this dictionary and pass it again on further
rendering requests; otherwise, you can just pass an empty dictionary.</param>
01607 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
01608 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
the largest page is used.</param>
01609 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
of the largest page is used.</param>
01610 /// <param name="background">The background colour of the document.  If this is <see langword="null"
/>, a transparent background is used.</param>
01611 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01612 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01613 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01614         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document,
Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string,
```

```
(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution = true, double?  width = null,
double?  height = null, Colour?  background = null, AvaloniaContextInterpreter.TextOptions textOption
= AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01615         {
01616             filterOption = filterOption ??  FilterOption.Default;
01617 
01618             return new SKMultiLayerRenderCanvas((from el in document.Pages select
el.CopyToSKRenderContext(taggedActions, images, removeTaggedActionsAfterExecution, textOption,
filterOption)).ToList(), (from el in document.Pages select
SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
select el.Height).Max());
01619         }
01620 
01621 /// <summary>
01622 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
renderer.
01623 /// </summary>
01624 /// <param name="page">The <see cref="Page"/> to render.</param>
01625 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01626 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01627 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01628         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page,
AvaloniaContextInterpreter.TextOptions textOption =
AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01629         {
01630             filterOption = filterOption ??  FilterOption.Default;
01631 
01632             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
page.CopyToSKRenderContext(textOption, filterOption) }, new List<SKRenderAction>() {
SKRenderAction.TransformAction(SKMatrix.Identity) }, page.Background, page.Width, page.Height);
01633         }
01634 
01635 /// <summary>
01636 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the
SkiaSharpRenderer.
01637 /// </summary>
01638 /// <param name="page">The <see cref="Page"/> to render.</param>
01639 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01640 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01641 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
01642 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01643 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01644 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01645         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page, Dictionary<string,
Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions textOption =
AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01646         {
01647             filterOption = filterOption ??  FilterOption.Default;
01648 
01649             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
page.CopyToSKRenderContext(taggedActions, removeTaggedActionsAfterExecution, textOption, filterOption)
}, new List<SKRenderAction>() { SKRenderAction.TransformAction(SKMatrix.Identity) }, page.Background,
page.Width, page.Height);
01650         }
01651 
01652 /// <summary>
01653 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the
SkiaSharpRenderer.
01654 /// </summary>
01655 /// <param name="page">The <see cref="Page"/> to render.</param>
01656 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01657 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01658 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
as "pixelated" or not.  This will be populated automatically as the page is rendered.
01659 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
times), it will be beneficial to keep a reference to this dictionary and pass it again on further
rendering requests; otherwise, you can just pass an empty dictionary.</param>
01660 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
```

```
01661 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01662 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01663 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01664         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string, (SKBitmap, bool)>
      images, bool removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions
      textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption =
      default)
01665         {
01666             filterOption = filterOption ??  FilterOption.Default;
01667 
01668             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
      page.CopyToSKRenderContext(taggedActions, images, removeTaggedActionsAfterExecution, textOption,
      filterOption) }, new List<SKRenderAction>() { SKRenderAction.TransformAction(SKMatrix.Identity) },
      page.Background, page.Width, page.Height);
01669         }
01670 
01671 /// <summary>
01672 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01673 /// </summary>
01674 /// <param name="page">The <see cref="Page"/> to render.</param>
01675 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01676 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01677 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01678         public static SKRenderContext CopyToSKRenderContext(this Page page,
      AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01679         {
01680             filterOption = filterOption ??  FilterOption.Default;
01681 
01682             return CopyToSKRenderContext(page, new Dictionary<string, Func<SKRenderAction,
      IEnumerable<SKRenderAction>>(), new Dictionary<string, (SKBitmap, bool)>(), textOption:  textOption,
      filterOption:  filterOption);
01683         }
01684 
01685 /// <summary>
01686 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01687 /// </summary>
01688 /// <param name="page">The <see cref="Page"/> to render.</param>
01689 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01690 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01691 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
      <paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
      performed on multiple items with the same tag.</param>
01692 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01693 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01694 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01695         public static SKRenderContext CopyToSKRenderContext(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
      removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01696         {
01697             filterOption = filterOption ??  FilterOption.Default;
01698 
01699             return CopyToSKRenderContext(page, taggedActions, new Dictionary<string, (SKBitmap,
      bool)>(), removeTaggedActionsAfterExecution, textOption, filterOption);
01700         }
01701 
01702 /// <summary>
01703 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01704 /// </summary>
01705 /// <param name="page">The <see cref="Page"/> to render.</param>
01706 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01707 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01708 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
      image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
      as "pixelated" or not.  This will be populated automatically as the page is rendered.
01709 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
      times), it will be beneficial to keep a reference to this dictionary and pass it again on further
      rendering requests; otherwise, you can just pass an empty dictionary.</param>
01710 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
```

```
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be performed on multiple items with the same tag.</param>
01711 /// <param name="textOption">Defines whether text items should be converted into paths when drawing.</param>
01712 /// <param name="filterOption">Defines how and whether image filters should be rasterised when rendering the image.</param>
01713 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01714         public static SKRenderContext CopyToSKRenderContext(this Page page, Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string, (SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01715         {
01716             filterOption = filterOption ??  FilterOption.Default;
01717 
01718             SKRenderContextImpl tbr = new SKRenderContextImpl(page.Width, page.Height, removeTaggedActionsAfterExecution, textOption, images, filterOption)
01719             {
01720                 TaggedActions = taggedActions
01721             };
01722             page.Graphics.CopyToIGraphicsContext(tbr);
01723 
01724             return tbr;
01725         }
01726 
01727 
01728 /// <summary>
01729 /// Rasterise a region of a <see cref="Graphics"/> object.
01730 /// </summary>
01731 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
01732 /// <param name="region">The region of the <paramref name="graphics"/> that will be rasterised.</param>
01733 /// <param name="scale">The scale at which the image will be rendered.</param>
01734 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is drawn on another <see cref="Graphics"/> surface.</param>
01735 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
01736         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale, bool interpolate)
01737         {
01738             Page pag = new Page(1, 1);
01739             pag.Graphics.DrawGraphics(0, 0, graphics);
01740             pag.Crop(region.Location, region.Size);
01741 
01742             //pag.PaintToSKCanvas().RenderAtResolution((int)Math.Round(region.Size.Width * scale), (int)Math.Round(region.Size.Height * scale), null);
01743 
01744             int width = (int)Math.Round(region.Size.Width * scale);
01745             int height = (int)Math.Round(region.Size.Height * scale);
01746 
01747             SKRenderContext ctx = pag.CopyToSKRenderContext();
01748 
01749             SKBitmap bitmap = new SKBitmap(width, height, SKColorType.Rgba8888, SKAlphaType.Unpremul);
01750 
01751             SKCanvas canvas = new SKCanvas(bitmap);
01752 
01753             canvas.Save();
01754 
01755             for (int i = 0; i < ctx.SKRenderActions.Count; i++)
01756             {
01757                 if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Clip)
01758                 {
01759                     canvas.ClipPath(ctx.SKRenderActions[i].Path, antialias:  true);
01760                 }
01761                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Restore)
01762                 {
01763                     canvas.Restore();
01764                 }
01765                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Save)
01766                 {
01767                     canvas.Save();
01768                 }
01769                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Transform)
01770                 {
01771                     SKMatrix mat = ctx.SKRenderActions[i].Transform.Value;
01772                     canvas.Concat(ref mat);
01773                 }
01774                 else
01775                 {
01776                     if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Path && ctx.SKRenderActions[i].Path != null)
01777                     {
01778                         canvas.DrawPath(ctx.SKRenderActions[i].Path, ctx.SKRenderActions[i].Paint);
01779                     }
01780                     else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Text)
01781                     {
01782                         canvas.DrawText(ctx.SKRenderActions[i].Text, ctx.SKRenderActions[i].TextX, ctx.SKRenderActions[i].TextY, ctx.SKRenderActions[i].Font, ctx.SKRenderActions[i].Paint);
```

```
01783                     }
01784                     else if (ctx.SKRenderActions[i].ActionType ==
    SKRenderAction.ActionTypes.RasterImage)
01785                     {
01786                         (SKBitmap image, bool interpolateIt) =
    ctx.Images[ctx.SKRenderActions[i].ImageId];
01787 
01788                         SKPaint paint;
01789 
01790                         if (!interpolateIt)
01791                         {
01792                             paint = null;
01793                         }
01794                         else
01795                         {
01796                             paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
01797                         }
01798 
01799                         canvas.DrawBitmap(image, ctx.SKRenderActions[i].ImageSource.Value,
    ctx.SKRenderActions[i].ImageDestination.Value, paint);
01800 
01801                         paint?.Dispose();
01802                     }
01803                 }
01804             }
01805 
01806             canvas.Restore();
01807 
01808             IntPtr pixels = bitmap.GetPixels(out IntPtr length);
01809 
01810             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(length);
01811             GC.AddMemoryPressure((long)length);
01812 
01813             unsafe
01814             {
01815                 Buffer.MemoryCopy((void*)pixels, (void*)tbrData, (long)length, (long)length);
01816             }
01817 
01818             canvas.Dispose();
01819             bitmap.Dispose();
01820 
01821             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
01822             return new RasterImage(ref disp, width, height, true, interpolate);
01823         }
01824     }
01825 }
```

8.7 Nimbus.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, version 3.
00008 This program is distributed in the hope that it will be useful,
00009 but WITHOUT ANY WARRANTY; without even the implied warranty of
00010 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00011 GNU General Public License for more details.
00012 You should have received a copy of the GNU General Public License
00013 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00014 */
00015 
00016 namespace VectSharp.Fonts
00017 {
00018 /// <summary>
00019 /// Contains an <see cref="IFontLibrary"/> providing access to the Nimbus family of standard fonts
    (used e.g.  by MuPDF).
00020 /// </summary>
00021     public class Nimbus
00022     {
00023 /// <summary>
00024 /// The font library.
00025 /// </summary>
00026         public static IFontLibrary Library { get; }
00027 
00028         static Nimbus()
00029         {
00030             Library = new SimpleFontLibrary(
00031                 new
    ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-Reg.t
    "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
```

```
00032                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-Med.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00033                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-RegIta
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00034                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-MedIta
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00035                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-Reg.ttf")
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00036                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-Bol.ttf")
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00037                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-RegIta.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00038                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-BolIta.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00039                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Regular.t
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00040                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Bold.ttf"
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00041                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Oblique.t
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00042                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-BoldOblic
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00043                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.StandardSymbolsPS.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#StandardSymbolsPS"),
00044                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.D050000L.ttf"),
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#D050000L"));
00045         }
00046     }
00047 }
```

8.8 HtmlTag.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Net;
00022 using System.Text;
00023 using System.Threading.Tasks;
00024
00025 namespace VectSharp.Markdown
00026 {
00027     internal class HtmlTag
00028     {
00029         public Dictionary<string, string> Attributes { get; }
00030         public string Tag { get; }
00031
00032         private HtmlTag(string tag, Dictionary<string, string> attributes)
00033         {
00034             this.Tag = tag;
00035             this.Attributes = attributes;
00036         }
00037
00038         private static HtmlTag ParseTag(StringReader reader)
```

```
00039         {
00040             StringBuilder tagBuilder = new StringBuilder();
00041 
00042             int character = reader.Read();
00043 
00044             while (character >= 0 && (char)character != '<')
00045             {
00046                 character = reader.Read();
00047             }
00048 
00049             if ((char)character == '<')
00050             {
00051                 character = reader.Read();
00052             }
00053 
00054             while (character >= 0 && char.IsWhiteSpace((char)character))
00055             {
00056                 character = reader.Read();
00057             }
00058 
00059             while (character >= 0 && !char.IsWhiteSpace((char)character) && (char)character != '>')
00060             {
00061                 tagBuilder.Append((char)character);
00062                 character = reader.Read();
00063             }
00064 
00065             string tag = tagBuilder.ToString();
00066 
00067             Dictionary<string, string> attributes = new Dictionary<string, string>();
00068 
00069             (string, string)?  attribute = ReadAttribute(reader, ref character);
00070 
00071             while (attribute != null && character >= 0)
00072             {
00073                 attributes[attribute.Value.Item1] = attribute.Value.Item2;
00074                 attribute = ReadAttribute(reader, ref character);
00075             }
00076 
00077             return new HtmlTag(tag, attributes);
00078         }
00079 
00080         public static IEnumerable<HtmlTag> ParseTagsUntil(StringReader reader, string targetTag)
00081         {
00082             HtmlTag tag = ParseTag(reader);
00083 
00084             while (tag.Tag != targetTag && reader.Peek() >= 0)
00085             {
00086                 if (tag.Tag.Equals("p", StringComparison.OrdinalIgnoreCase))
00087                 {
00088                     foreach (HtmlTag nestedTag in ParseTagsUntil(reader, "/p"))
00089                     {
00090                         if (nestedTag.Tag != "/p")
00091                         {
00092                             foreach (KeyValuePair<string, string> kvp in tag.Attributes)
00093                             {
00094                                 if (!nestedTag.Attributes.ContainsKey(kvp.Key))
00095                                 {
00096                                     nestedTag.Attributes[kvp.Key] = kvp.Value;
00097                                 }
00098                             }
00099 
00100                             yield return nestedTag;
00101                         }
00102                     }
00103                 }
00104                 else
00105                 {
00106                     yield return tag;
00107                 }
00108 
00109 
00110                 tag = ParseTag(reader);
00111             }
00112 
00113             yield return tag;
00114         }
00115 
00116         public static IEnumerable<HtmlTag> Parse(string html)
00117         {
00118             using (StringReader reader = new StringReader(html))
00119             {
00120                 while (reader.Peek() >= 0)
00121                 {
00122                     HtmlTag tag = ParseTag(reader);
00123 
00124                     if (tag.Tag.Equals("p", StringComparison.OrdinalIgnoreCase))
00125                     {
```

```
00126                         foreach (HtmlTag nestedTag in ParseTagsUntil(reader, "/p"))
00127                         {
00128                             if (nestedTag.Tag != "/p")
00129                             {
00130                                 foreach (KeyValuePair<string, string> kvp in tag.Attributes)
00131                                 {
00132                                     if (!nestedTag.Attributes.ContainsKey(kvp.Key))
00133                                     {
00134                                         nestedTag.Attributes[kvp.Key] = kvp.Value;
00135                                     }
00136                                 }
00137 
00138                                 yield return nestedTag;
00139                             }
00140                         }
00141                     }
00142                     else
00143                     {
00144                         yield return tag;
00145                     }
00146                 }
00147             }
00148         }
00149 
00150         private static (string, string)?  ReadAttribute(StringReader reader, ref int character)
00151         {
00152             while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character != '>')
00153             {
00154                 character = reader.Read();
00155             }
00156 
00157             if ((char)character == '>')
00158             {
00159                 return null;
00160             }
00161             else
00162             {
00163                 StringBuilder attributeNameBuilder = new StringBuilder();
00164 
00165                 while (character >= 0 && !char.IsWhiteSpace((char)character) && (char)character != '>'
      && (char)character != '=')
00166                 {
00167                     attributeNameBuilder.Append((char)character);
00168                     character = reader.Read();
00169                 }
00170 
00171                 string attributeName = attributeNameBuilder.ToString();
00172 
00173                 while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character != '>'
      && (char)character != '=')
00174                 {
00175                     character = reader.Read();
00176                 }
00177 
00178                 if ((char)character == '=')
00179                 {
00180                     character = reader.Read();
00181 
00182                     while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character !=
      '>')
00183                     {
00184                         character = reader.Read();
00185                     }
00186 
00187                     if ((char)character == '>')
00188                     {
00189                         return (attributeName, null);
00190                     }
00191                     else
00192                     {
00193                         bool quoted = (char)character == '"' || (char)character == '\'';
00194 
00195                         if (quoted)
00196                         {
00197                             char quoteChar = (char)character;
00198 
00199                             character = reader.Read();
00200 
00201                             StringBuilder attributeValueBuilder = new StringBuilder();
00202 
00203                             bool isEscaped = (char)character == '\\';
00204 
00205                             while (character >= 0 && ((char)character != quoteChar || isEscaped))
00206                             {
00207                                 attributeValueBuilder.Append((char)character);
00208                                 character = reader.Read();
00209                                 isEscaped = (char)character == '\\' && !isEscaped;
```

```
00210                               }
00211
00212                               string attributeValue = attributeValueBuilder.ToString();
00213
00214                               return (attributeName, attributeValue);
00215                           }
00216                           else
00217                           {
00218                               StringBuilder attributeValueBuilder = new StringBuilder();
00219
00220                               while (character >= 0 && !char.IsWhiteSpace((char)character) &&
      (char)character != '>' && (char)character != '=')
00221                               {
00222                                   attributeValueBuilder.Append((char)character);
00223                                   character = reader.Read();
00224                               }
00225
00226                               string attributeValue = attributeValueBuilder.ToString();
00227
00228                               return (attributeName, attributeValue);
00229                           }
00230                       }
00231                   }
00232                   else
00233                   {
00234                       return (attributeName, null);
00235                   }
00236               }
00237           }
00238
00239       }
00240
00241 /// <summary>
00242 /// Contains utilities to resolve absolute and relative URIs.
00243 /// </summary>
00244       public static class HTTPUtils
00245       {
00246 /// <summary>
00247 /// Determines whether every file that is downloaded should be logged to the standard error stream.
00248 /// </summary>
00249           public static bool LogDownloads { get; set; } = true;
00250
00251 /// <summary>
00252 /// Resolves an image Uri, by downloading the image file if necessary.  It also takes care of ensuring
      that the file extension matches the format of the file.
00253 /// </summary>
00254 /// <param name="uri">The address of the image.</param>
00255 /// <param name="baseUriString">The base uri to use for relative uris.</param>
00256 /// <returns>A tuple containing the local path of the image file (either the original image, or a
      local copy of a remote file) and a boolean value indicating whether the image was fetched from a
      remote location and should be deleted after the program is done with it.</returns>
00257           public static (string path, bool wasDownloaded) ResolveImageURI(string uri, string
      baseUriString)
00258           {
00259               if (uri.StartsWith("data:"))
00260               {
00261                   string tempFile = Path.GetTempFileName();
00262                   if (File.Exists(tempFile))
00263                   {
00264                       File.Delete(tempFile);
00265                   }
00266
00267                   Directory.CreateDirectory(tempFile);
00268
00269                   if (!uri.StartsWith(""))
00270                   {
00271                       VectSharp.Page pag = VectSharp.SVG.Parser.ParseImageURI(uri, true);
00272                       VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, Path.Combine(tempFile,
      "temp.svg"));
00273                   }
00274                   else
00275                   {
00276                       string base64 = uri.Substring("".Length);
00277
00278                       File.WriteAllBytes(Path.Combine(tempFile, "temp.svg"),
      Convert.FromBase64String(base64));
00279                   }
00280
00281                   return (Path.Combine(tempFile, "temp.svg"), true);
00282               }
00283               else if (File.Exists(Path.Combine(baseUriString, uri)))
00284               {
00285                   return (Path.Combine(baseUriString, uri), false);
00286               }
00287               else if (File.Exists(uri))
00288               {
00289                   return (uri, false);
```

```
00290             }
00291             else
00292             {
00293                 Uri absoluteUri;
00294                 bool validUri;
00295 
00296                 if (Uri.TryCreate(baseUriString, UriKind.Absolute, out Uri baseUri))
00297                 {
00298                     validUri = Uri.TryCreate(baseUri, uri, out absoluteUri);
00299                 }
00300                 else
00301                 {
00302                     validUri = Uri.TryCreate(uri, UriKind.Absolute, out absoluteUri);
00303                 }
00304 
00305                 if (validUri)
00306                 {
00307                     string tempFile = Path.GetTempFileName();
00308                     File.Delete(tempFile);
00309                     Directory.CreateDirectory(tempFile);
00310 
00311                     string fileDest = Path.Combine(tempFile, Path.GetFileName(absoluteUri.LocalPath));
00312 
00313                     try
00314                     {
00315                         if (LogDownloads)
00316                         {
00317                             Console.Error.WriteLine();
00318                             Console.Error.Write("Downloading {0}...", absoluteUri);
00319                         }
00320 
00321                         using (WebClient client = new WebClient())
00322                         {
00323                             client.DownloadFile(absoluteUri, fileDest);
00324                         }
00325 
00326                         if (LogDownloads)
00327                         {
00328                             Console.Error.WriteLine(" Done.");
00329                         }
00330 
00331                         string newName = FixFileExtensionBasedOnContent(fileDest);
00332 
00333                         File.Move(fileDest, newName);
00334                         fileDest = newName;
00335 
00336                         return (fileDest, true);
00337                     }
00338                     catch (Exception ex)
00339                     {
00340                         if (LogDownloads)
00341                         {
00342                             Console.Error.WriteLine(" Failed!");
00343                             Console.Error.WriteLine(ex.Message);
00344                         }
00345 
00346                         Directory.Delete(tempFile, true);
00347                         return (null, false);
00348                     }
00349                 }
00350                 else
00351                 {
00352                     return (null, false);
00353                 }
00354             }
00355         }
00356 
00357         private static string FixFileExtensionBasedOnContent(string fileName)
00358         {
00359             using (FileStream fileStream = File.OpenRead(fileName))
00360             {
00361                 bool isSvg = false;
00362 
00363                 try
00364                 {
00365                     using (var xmlReader = System.Xml.XmlReader.Create(fileStream))
00366                     {
00367                         isSvg = xmlReader.MoveToContent() == System.Xml.XmlNodeType.Element &&
    "svg".Equals(xmlReader.Name, StringComparison.OrdinalIgnoreCase);
00368                     }
00369                 }
00370                 catch
00371                 {
00372                     isSvg = false;
00373                 }
00374 
00375                 if (isSvg)
```

```
00376                 {
00377                     return fileName + ".svg";
00378                 }
00379                 else
00380                 {
00381                     fileStream.Seek(0, SeekOrigin.Begin);
00382                     byte[] header = new byte[8];
00383 
00384                     for (int i = 0; i < header.Length; i++)
00385                     {
00386                         header[i] = (byte)fileStream.ReadByte();
00387                     }
00388 
00389                     if (header[0] == 0x42 && header[1] == 0x4D)
00390                     {
00391                         return fileName + ".bmp";
00392                     }
00393                     else if (header[0] == 0x47 && header[1] == 0x49 && header[2] == 0x46 && header[3]
      == 0x38)
00394                     {
00395                         return fileName + ".gif";
00396                     }
00397                     else if (header[0] == 0xFF && header[1] == 0xD8 && header[2] == 0xFF && (header[3]
      == 0xDB || header[3] == 0xE0 || header[3] == 0xEE || header[3] == 0xE1))
00398                     {
00399                         return fileName + ".jpg";
00400                     }
00401                     else if (header[0] == 0x25 && header[1] == 0x50 && header[2] == 0x44 && header[3]
      == 0x46 && header[4] == 0x2D)
00402                     {
00403                         return fileName + ".pdf";
00404                     }
00405                     else if (header[0] == 0x89 && header[1] == 0x50 && header[2] == 0x4E && header[3]
      == 0x47 && header[4] == 0x0D && header[5] == 0x0A && header[6] == 0x1A && header[7] == 0x0A)
00406                     {
00407                         return fileName + ".png";
00408                     }
00409                     else if ((header[0] == 0x49 && header[1] == 0x49 && header[2] == 0x2A && header[3]
      == 0x00) || (header[0] == 0x4D && header[1] == 0x4D && header[2] == 0x00 && header[3] == 0x2A))
00410                     {
00411                         return fileName + ".tif";
00412                     }
00413                     else
00414                     {
00415                         return fileName;
00416                     }
00417                 }
00418             }
00419         }
00420     }
00421 }
```

8.9 Line.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021 
00022 namespace VectSharp.Markdown
00023 {
00024     internal abstract class LineFragment
00025     {
00026         public abstract void Translate(double deltaX, double deltaY);
00027         public string Tag { get; protected set; }
00028     }
00029 
```

```
00030     internal class TextFragment :  LineFragment
00031     {
00032         public string Text { get; }
00033         public Point Position { get; private set; }
00034         public Font Font { get; }
00035         public Colour Colour { get; }
00036 
00037         public TextFragment(Point position, string text, Font font, Colour colour, string tag)
00038         {
00039             this.Position = position;
00040             this.Text = text;
00041             this.Font = font;
00042             this.Colour = colour;
00043             this.Tag = tag;
00044         }
00045 
00046         public override void Translate(double deltaX, double deltaY)
00047         {
00048             this.Position = new Point(this.Position.X + deltaX, this.Position.Y + deltaY);
00049         }
00050     }
00051 
00052     internal class UnderlineFragment :  LineFragment
00053     {
00054         public Point Start { get; private set; }
00055         public Point End { get; private set; }
00056         public Colour Colour { get; }
00057         public double Thickness { get; }
00058 
00059         public UnderlineFragment(Point start, Point end, Colour colour, double thickness, string tag)
00060         {
00061             this.Start = start;
00062             this.End = end;
00063             this.Colour = colour;
00064             this.Thickness = thickness;
00065             this.Tag = tag;
00066         }
00067 
00068         public override void Translate(double deltaX, double deltaY)
00069         {
00070             this.Start = new Point(this.Start.X + deltaX, this.Start.Y + deltaY);
00071             this.End = new Point(this.End.X + deltaX, this.End.Y + deltaY);
00072         }
00073     }
00074 
00075     internal class RectangleFragment :  LineFragment
00076     {
00077         public Point TopLeft { get; private set; }
00078         public Size Size { get; }
00079         public Colour Colour { get; }
00080 
00081         public RectangleFragment(Point topLeft, Size size, Colour colour, string tag)
00082         {
00083             this.TopLeft = topLeft;
00084             this.Size = size;
00085             this.Colour = colour;
00086             this.Tag = tag;
00087         }
00088 
00089         public override void Translate(double deltaX, double deltaY)
00090         {
00091             this.TopLeft = new Point(this.TopLeft.X + deltaX, this.TopLeft.Y + deltaY);
00092         }
00093     }
00094 
00095     internal class GraphicsFragment :  LineFragment
00096     {
00097         public Point Origin { get; private set; }
00098         public Graphics Graphics { get; }
00099         public double Ascent { get; }
00100 
00101         public GraphicsFragment(Point origin, Graphics graphics, double ascent)
00102         {
00103             this.Origin = origin;
00104             this.Graphics = graphics;
00105             this.Ascent = ascent;
00106         }
00107 
00108         public override void Translate(double deltaX, double deltaY)
00109         {
00110             this.Origin = new Point(this.Origin.X + deltaX, this.Origin.Y + deltaY);
00111         }
00112     }
00113 
00114     internal class Line
00115     {
00116         public List<LineFragment> Fragments { get; }
```

```
00117         public double InitialAscent { get; }
00118
00119         public Line(double initialAscent)
00120         {
00121             this.InitialAscent = initialAscent;
00122             this.Fragments = new List<LineFragment>();
00123         }
00124
00125         public void Render(ref Graphics graphics, ref MarkdownContext context,
      MarkdownRenderer.NewPageAction newPageAction, double pageMaxY)
00126         {
00127             double deltaY = 0;
00128             double maxY = 0;
00129
00130             foreach (LineFragment fragment in this.Fragments)
00131             {
00132                 if (fragment is TextFragment text)
00133                 {
00134                     deltaY = Math.Max(deltaY, text.Font.Ascent - this.InitialAscent);
00135                     maxY = Math.Max(maxY, text.Position.Y - text.Font.Descent);
00136                 }
00137                 else if (fragment is UnderlineFragment underline)
00138                 {
00139                     maxY = Math.Max(maxY, Math.Max(underline.Start.Y, underline.End.Y));
00140                 }
00141                 else if (fragment is RectangleFragment rectangle)
00142                 {
00143                     maxY = Math.Max(maxY, rectangle.TopLeft.Y + rectangle.Size.Height);
00144                 }
00145                 else if (fragment is GraphicsFragment graphicsFragment)
00146                 {
00147                     deltaY = Math.Max(deltaY, graphicsFragment.Ascent - this.InitialAscent);
00148                     maxY = Math.Max(maxY, graphicsFragment.Origin.Y);
00149                 }
00150             }
00151
00152             maxY += deltaY;
00153
00154             if (maxY > pageMaxY)
00155             {
00156                 double currCursY = context.Cursor.Y;
00157
00158                 newPageAction(ref context, ref graphics);
00159
00160                 double currDelta = deltaY;
00161
00162                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + this.InitialAscent +
      currDelta);
00163
00164                 deltaY -= currCursY - context.Cursor.Y + currDelta;
00165
00166                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y);
00167             }
00168             else
00169             {
00170                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + deltaY);
00171             }
00172
00173
00174             for (int i = 0; i < this.Fragments.Count; i++)
00175             {
00176                 LineFragment fragment = this.Fragments[i];
00177                 if (fragment is TextFragment text)
00178                 {
00179                     Size size = text.Font.MeasureText(text.Text);
00180                     context.BottomRight = new Point(Math.Max(context.BottomRight.X, size.Width +
      text.Position.X + context.Translation.X), Math.Max(context.BottomRight.Y, text.Position.Y +
      size.Height + deltaY + context.Translation.Y));
00181                     graphics.FillText(text.Position.X, text.Position.Y + deltaY, text.Text, text.Font,
      text.Colour, TextBaselines.Baseline, tag:  fragment.Tag);
00182                 }
00183                 else if (fragment is UnderlineFragment underline)
00184                 {
00185                     graphics.StrokePath(new GraphicsPath().MoveTo(underline.Start.X, underline.Start.Y
      + deltaY).LineTo(underline.End.X, underline.End.Y + deltaY), underline.Colour, underline.Thickness,
      tag:  fragment.Tag);
00186                 }
00187                 else if (fragment is RectangleFragment rectangle)
00188                 {
00189                     graphics.FillRectangle(rectangle.TopLeft.X, rectangle.TopLeft.Y + deltaY,
      rectangle.Size.Width, rectangle.Size.Height, rectangle.Colour, tag:  fragment.Tag);
00190                 }
00191                 else if (fragment is GraphicsFragment graphicsFragment)
00192                 {
00193                     graphics.DrawGraphics(graphicsFragment.Origin.X, graphicsFragment.Origin.Y +
      deltaY, graphicsFragment.Graphics);
00194                 }
```

```
00195             }
00196         }
00197     }
00198 }
```

8.10 MarkdownContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021
00022 namespace VectSharp.Markdown
00023 {
00024     internal class MarkdownContext
00025     {
00026         public Font Font { get; set; }
00027
00028         private Point cursor;
00029         public Point Cursor
00030         {
00031             get
00032             {
00033                 return cursor;
00034             }
00035
00036             set
00037             {
00038                 cursor = value;
00039
00040                 double maxX = Math.Max(BottomRight.X, value.X + Translation.X);
00041                 double maxY = Math.Max(BottomRight.Y, value.Y + Translation.Y);
00042
00043                 this.BottomRight = new Point(maxX, maxY);
00044             }
00045         }
00046
00047         public Colour Colour { get; set; }
00048         public bool Underline { get; set; }
00049         public bool StrikeThrough { get; set; }
00050
00051         private Point translation;
00052         public Point Translation
00053         {
00054             get
00055             {
00056                 return translation;
00057             }
00058
00059             set
00060             {
00061                 translation = value;
00062
00063                 double maxX = Math.Max(BottomRight.X, cursor.X + value.X);
00064                 double maxY = Math.Max(BottomRight.Y, cursor.Y + value.Y);
00065
00066                 this.BottomRight = new Point(maxX, maxY);
00067             }
00068         }
00069
00070         public Point MarginBottomRight { get; set; }
00071
00072         public Line CurrentLine { get; set; }
00073         public int ListDepth { get; set; }
00074         public List<(double MaxX, double MinY, double MaxY)> ForbiddenAreasRight { get; set; }
00075         public List<(double MinX, double MinY, double MaxY)> ForbiddenAreasLeft { get; set; }
00076         public Page CurrentPage { get; set; }
```

```
00077         public Point BottomRight { get; set; }
00078         public string Tag { get; set; } = null;
00079         public Dictionary<string, string> LinkDestinations { get; set; } = new Dictionary<string,
      string>();
00080         public Dictionary<string, string> InternalAnchors { get; set; } = new Dictionary<string,
      string>();
00081
00082         public MarkdownContext()
00083         {
00084             this.ForbiddenAreasRight = new List<(double MaxX, double MinY, double MaxY)>();
00085             this.ForbiddenAreasLeft = new List<(double MinX, double MinY, double MaxY)>();
00086         }
00087
00088         public MarkdownContext Clone()
00089         {
00090             return new MarkdownContext()
00091             {
00092                 Font = this.Font,
00093                 Cursor = this.Cursor,
00094                 Colour = this.Colour,
00095                 Underline = this.Underline,
00096                 StrikeThrough = this.StrikeThrough,
00097                 Translation = this.Translation,
00098                 CurrentLine = this.CurrentLine,
00099                 ListDepth = this.ListDepth,
00100                 ForbiddenAreasRight = this.ForbiddenAreasRight,
00101                 ForbiddenAreasLeft = this.ForbiddenAreasLeft,
00102                 CurrentPage = this.CurrentPage,
00103                 Tag = this.Tag,
00104                 LinkDestinations = this.LinkDestinations,
00105                 InternalAnchors = this.InternalAnchors,
00106                 MarginBottomRight = this.MarginBottomRight
00107             };
00108         }
00109
00110         public double GetMaxX(double y, double pageMaxX)
00111         {
00112             y = y + Translation.Y;
00113
00114             double maxX = pageMaxX;
00115
00116             foreach ((double MaxX, double MinY, double MaxY) in ForbiddenAreasRight)
00117             {
00118                 if (MinY <= y && MaxY >= y)
00119                 {
00120                     maxX = Math.Min(maxX, MaxX - this.Translation.X);
00121                 }
00122             }
00123
00124             return maxX;
00125         }
00126
00127         public double GetMaxX(double y0, double y1, double pageMaxX)
00128         {
00129             y0 = y0 + Translation.Y;
00130             y1 = y1 + Translation.Y;
00131
00132             double maxX = pageMaxX;
00133
00134             foreach ((double MaxX, double MinY, double MaxY) in ForbiddenAreasRight)
00135             {
00136                 if (!((MinY < y0 && MaxY < y0) || (MinY > y1 && MaxY > y1)))
00137                 {
00138                     maxX = Math.Min(maxX, MaxX - this.Translation.X);
00139                 }
00140             }
00141
00142             return maxX;
00143         }
00144
00145         public double GetMinX(double y)
00146         {
00147             y = y + Translation.Y;
00148
00149             double minX = 0;
00150
00151             foreach ((double MinX, double MinY, double MaxY) in ForbiddenAreasLeft)
00152             {
00153                 if (MinY <= y && MaxY >= y)
00154                 {
00155                     minX = Math.Max(minX, MinX - Translation.X);
00156                 }
00157             }
00158
00159             return minX;
00160         }
00161
```

```
00162         public double GetMinX(double y0, double y1)
00163         {
00164 
00165             y0 = y0 + Translation.Y;
00166             y1 = y1 + Translation.Y;
00167 
00168             double minX = 0;
00169 
00170             foreach ((double MinX, double MinY, double MaxY) in ForbiddenAreasLeft)
00171             {
00172                 if (!((MinY < y0 && MaxY < y0) || (MinY > y1 && MaxY > y1)))
00173                 {
00174                     minX = Math.Max(minX, MinX - Translation.X);
00175                 }
00176             }
00177 
00178             return minX;
00179         }
00180     }
00181 
00182 /// <summary>
00183 /// Represents the margins of a page.
00184 /// </summary>
00185     public class Margins
00186     {
00187 /// <summary>
00188 /// The left margin.
00189 /// </summary>
00190         public double Left { get; }
00191 
00192 /// <summary>
00193 /// The right margin.
00194 /// </summary>
00195         public double Right { get; }
00196 
00197 /// <summary>
00198 /// The top margin.
00199 /// </summary>
00200         public double Top { get; }
00201 
00202 /// <summary>
00203 /// The bottom margin.
00204 /// </summary>
00205         public double Bottom { get; }
00206 
00207 /// <summary>
00208 /// Creates a new <see cref="Margins"/> instance.
00209 /// </summary>
00210 /// <param name="left">The left margin.</param>
00211 /// <param name="top">The top margin.</param>
00212 /// <param name="right">The right margin.</param>
00213 /// <param name="bottom">The bottom margin.</param>
00214         public Margins(double left, double top, double right, double bottom)
00215         {
00216             this.Left = left;
00217             this.Right = right;
00218             this.Top = top;
00219             this.Bottom = bottom;
00220         }
00221     }
00222 }
```

8.11 MarkdownRenderer.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using VectSharp.Markdown.CSharpMath.VectSharp;
00019 using Markdig;
```

```
00020 using Markdig.Extensions;
00021 using Markdig.Extensions.Tables;
00022 using Markdig.Helpers;
00023 using Markdig.Renderers.Html;
00024 using Markdig.Syntax;
00025 using Markdig.Syntax.Inlines;
00026 using System;
00027 using System.Collections.Generic;
00028 using System.IO;
00029 using System.Linq;
00030 using System.Text;
00031
00032 namespace VectSharp.Markdown
00033 {
00034 /// <summary>
00035 /// Renders Markdown documents into VectSharp graphics objects.
00036 /// </summary>
00037     public class MarkdownRenderer
00038     {
00039 /// <summary>
00040 /// The base font size to use when rendering the document.  This will be the size of regular elements, and the size of header elements will be expressed as a multiple of this.
00041 /// </summary>
00042         public double BaseFontSize { get; set; } = 9.71424;
00043
00044 /// <summary>
00045 /// Scaling factor for the font size to use when rendering math.
00046 /// </summary>
00047         public double MathFontScalingFactor { get; set; } = 0.85;
00048
00049 /// <summary>
00050 /// The font size for elements at each header level.  The values in this array will be multiplied by the <see cref="BaseFontSize"/>.
00051 /// </summary>
00052         public double[] HeaderFontSizeMultipliers { get; } = new double[]
00053         {
00054             28 / 12.0, 22 / 12.0, 16 / 12.0, 14 / 12.0, 13 / 12.0, 12 / 12.0
00055         };
00056
00057 /// <summary>
00058 /// The thickness of the separator line after a header of each level.  A value of 0 disables the line after headers of that level.
00059 /// </summary>
00060         public double[] HeaderLineThicknesses { get; } = new double[] { 1, 1, 0, 0, 0, 0 };
00061
00062 /// <summary>
00063 /// The thickness of thematic break lines.
00064 /// </summary>
00065         public double ThematicBreakThickness { get; set; } = 2;
00066
00067 /// <summary>
00068 /// The font family for regular text.
00069 /// </summary>
00070         public FontFamily RegularFontFamily { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00071
00072 /// <summary>
00073 /// The font family for bold text.
00074 /// </summary>
00075         public FontFamily BoldFontFamily { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold);
00076
00077 /// <summary>
00078 /// The font family for italic text.
00079 /// </summary>
00080         public FontFamily ItalicFontFamily { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00081
00082 /// <summary>
00083 /// The font family for bold italic text.
00084 /// </summary>
00085         public FontFamily BoldItalicFontFamily { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00086
00087 /// <summary>
00088 /// The font family for code elements.
00089 /// </summary>
00090         public FontFamily CodeFont { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00091
00092 /// <summary>
00093 /// The font family for bold code elements.
00094 /// </summary>
00095         public FontFamily CodeFontBold { get; set; } = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold);
00096
00097 /// <summary>
```

```
00098 /// The font family for italic code elements.
00099 /// </summary>
00100         public FontFamily CodeFontItalic { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00101
00102 /// <summary>
00103 /// The font family for bold italic code elements.
00104 /// </summary>
00105         public FontFamily CodeFontBoldItalic { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00106
00107 /// <summary>
00108 /// The thickness of underlines.  This value will be multiplied by the font size of the element being
      underlined.
00109 /// </summary>
00110         public double UnderlineThickness { get; set; } = 0.075;
00111
00112 /// <summary>
00113 /// The thickness of underlines for bold text.  This value will be multiplied by the font size of the
      element being underlined.
00114 /// </summary>
00115         public double BoldUnderlineThickness { get; set; } = 0.15;
00116
00117 /// <summary>
00118 /// The margins of the page.
00119 /// </summary>
00120         public Margins Margins { get; set; } = new Margins(55, 55, 55, 55);
00121
00122 /// <summary>
00123 /// The margins for table cells.
00124 /// </summary>
00125         public Margins TableCellMargins { get; set; } = new Margins(5, 0, 5, 0);
00126
00127 /// <summary>
00128 /// Defines the options for the vertical alignment of table cells.
00129 /// </summary>
00130         public enum VerticalAlignment
00131         {
00132 /// <summary>
00133 /// Table cells will be aligned at the top of their row.
00134 /// </summary>
00135             Top,
00136
00137 /// <summary>
00138 /// Table cells will be aligned in the middle of their row.
00139 /// </summary>
00140             Middle,
00141
00142 /// <summary>
00143 /// Table cells will be aligned at the bottom of their row.
00144 /// </summary>
00145             Bottom
00146         }
00147
00148 /// <summary>
00149 /// The vertical alignment of table cells.
00150 /// </summary>
00151         public VerticalAlignment TableVAlign { get; set; } = VerticalAlignment.Middle;
00152
00153 /// <summary>
00154 /// The size of the page.
00155 /// </summary>
00156         public Size PageSize { get; set; } = new Size(595, 842);
00157
00158 /// <summary>
00159 /// The space before each text paragraph.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00160 /// </summary>
00161         public double SpaceBeforeParagaph { get; set; } = 0;
00162
00163 /// <summary>
00164 /// The space after each text paragraph.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00165 /// </summary>
00166         public double SpaceAfterParagraph { get; set; } = 0.75;
00167
00168 /// <summary>
00169 /// The space after each line of text.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00170 /// </summary>
00171         public double SpaceAfterLine { get; set; } = 0.25;
00172
00173 /// <summary>
00174 /// The space before each heading.  This value will be multiplied by the font size of the heading.
00175 /// </summary>
00176         public double SpaceBeforeHeading { get; set; } = 0.25;
00177
```

```
00178 /// <summary>
00179 /// The space after each heading.  This value will be multiplied by the font size of the heading.
00180 /// </summary>
00181         public double SpaceAfterHeading { get; set; } = 0.25;
00182
00183 /// <summary>
00184 /// The margin at the left and right of code inlines.  This value will be multiplied by the current
      font size.
00185 /// </summary>
00186         public double CodeInlineMargin { get; set; } = 0.25;
00187
00188 /// <summary>
00189 /// The indentation width used for list items.
00190 /// </summary>
00191         public double IndentWidth { get; set; } = 40;
00192
00193 /// <summary>
00194 /// The indentation width used for block quotes.
00195 /// </summary>
00196         public double QuoteBlockIndentWidth { get; set; } = 30;
00197
00198 /// <summary>
00199 /// The thickness of the bar to the left of block quotes.
00200 /// </summary>
00201         public double QuoteBlockBarWidth { get; set; } = 5;
00202
00203 /// <summary>
00204 /// The font size for subscripts and superscripts.  This value will be multiplied by the current font
      size.
00205 /// </summary>
00206         public double SubSuperscriptFontSize { get; set; } = 0.7;
00207
00208 /// <summary>
00209 /// The upwards shift in the baseline for superscript elements.  This value will be multiplied by the
      current font size.
00210 /// </summary>
00211         public double SuperscriptShift { get; set; } = 0.33;
00212
00213 /// <summary>
00214 /// The downwards shift in the baseline for subscript elements.  This value will be multiplied by the
      current font size.
00215 /// </summary>
00216         public double SubscriptShift { get; set; } = 0.14;
00217
00218 /// <summary>
00219 /// The base uri for resolving relative image addresses.
00220 /// </summary>
00221         public string BaseImageUri { get; set; } = "";
00222
00223 /// <summary>
00224 /// A method used to resolve (possibly remote) image uris into local file paths.  The first argument
      of the method should be the image uri and the second argument the base uri used to resolve relative
      links.  The method should return a tuple containing the path of the local file and a boolean value
      indicating whether the file has been fetched from a remote location and should be deleted after the
      program has finished using it.
00225 /// </summary>
00226         public Func<string, string, (string, bool)> ImageUriResolver { get; set; } =
      HTTPUtils.ResolveImageURI;
00227
00228 /// <summary>
00229 /// The base uri for resolving links.
00230 /// </summary>
00231         public Uri BaseLinkUri { get; set; } = new Uri("about:blank");
00232
00233 /// <summary>
00234 /// A method used to resolve link addresses.  The argument of the method should be the absolute link,
      and the method should return the resolved address.  This can be used to "redirect" links to a
      different target.
00235 /// </summary>
00236         public Func<string, string> LinkUriResolver { get; set; } = a => a;
00237
00238 /// <summary>
00239 /// A method used to a load raster image from a local file.  The argument of the method should be the
      path of a local image file, and the method should return a RasterImage representing that file.  For
      example, this can be achieved using the <c>RasterImageFile</c> class from the
      <c>VectSharp.MuPDFUtils</c> package.  If this is <see langword="null" />, only SVG images will be
      included in the document.
00240 /// </summary>
00241         public Func<string, RasterImage> RasterImageLoader { get; set; } = null;
00242
00243 /// <summary>
00244 /// The size of images (as defined in the image's width and height attributes) will be multiplied by
      this value to determine the actual size of the image on the page.  This has no effect on images
      without a width or height attribute.
00245 /// </summary>
00246         public double ImageUnitMultiplier { get; set; } = 0.60714;
00247
```

```
00248 /// <summary>
00249 /// The size of images will be multiplied by this value to determine the actual size of the image on
      the page.  For images that have a width or height attribute, this will be applied in addition to the
      <see cref="ImageUnitMultiplier"/>.  For images without width and height, only this multiplier will be
      applied.
00250 /// </summary>
00251         public double ImageMultiplier { get; set; } = 1;
00252
00253 /// <summary>
00254 /// The margin on the right of left-aligned images and on the left of right-aligned images.
00255 /// </summary>
00256         public double ImageSideMargin { get; set; } = 10;
00257
00258 /// <summary>
00259 /// Images will be allowed to extend into the page bottom margin area by this amount before triggering
      a page break.  This should be smaller than the bottom margin, otherwise images risk being cut off by
      the page boundary.
00260 /// </summary>
00261         public double ImageMarginTolerance { get; set; } = 25;
00262
00263 /// <summary>
00264 /// A method used for syntax highlighting.  The first argument should be the source code to highlight,
      while the second parameter is the name of the language to use for the highlight.  The method should
      return a list of lists of <see cref="FormattedString"/>s, with each list of <see
      cref="FormattedString"/>s representing a line.  For each code block, if the method returns <see
      langword="null" />, no syntax highlighting is used.
00265 /// </summary>
00266         public Func<string, string, List<List<FormattedString>> SyntaxHighlighter { get; set; } =
      VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlightedLines;
00267
00268 /// <summary>
00269 /// Bullet points used for unordered lists.  Each element of this list corresponds to the bullet for
      each level of list indentation.  If the list indentation is greater than the number of elements in
      this list, the bullet points will be reused cyclically.
00270 /// Each element of this list is a method taking two arguments:  the first is the <see
      cref="Graphics"/> object on which the bullet point should be drawn, while the second is the colour in
      which it should be painted.  The method should draw the bullet point centered around the origin.  The
      size of the bullet point will be multiplied by the current font size.
00271 /// </summary>
00272         public List<Action<Graphics, Colour» Bullets { get; } = new List<Action<Graphics, Colour»()
00273         {
00274             (graphics, colour) =>
00275             {
00276                 graphics.FillPath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour);
00277             },
00278
00279             (graphics, colour) =>
00280             {
00281                 graphics.StrokePath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour,
      0.1);
00282             },
00283
00284             (graphics, colour) =>
00285             {
00286                 graphics.StrokeRectangle(-0.75, -0.25, 0.5, 0.5, colour, 0.1);
00287             },
00288         };
00289
00290 /// <summary>
00291 /// The foreground colour for text elements.
00292 /// </summary>
00293         public Colour ForegroundColour { get; set; } = Colours.Black;
00294
00295 /// <summary>
00296 /// The background colour for the page.
00297 /// </summary>
00298         public Colour BackgroundColour { get; set; } = Colours.White;
00299
00300 /// <summary>
00301 /// The colour of the line below headers.
00302 /// </summary>
00303         public Colour HeaderLineColour { get; set; } = Colour.FromRgb(180, 180, 180);
00304
00305 /// <summary>
00306 /// The colour for thematic break lines.
00307 /// </summary>
00308         public Colour ThematicBreakLineColour { get; set; } = Colour.FromRgb(180, 180, 200);
00309
00310 /// <summary>
00311 /// The colour for hypertext links-
00312 /// </summary>
00313         public Colour LinkColour { get; set; } = Colour.FromRgb(25, 140, 191);
00314
00315 /// <summary>
00316 /// The background colour for code inlines.
00317 /// </summary>
00318         public Colour CodeInlineBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 240);
```

```
00319
00320 /// <summary>
00321 /// The background colour for code blocks.
00322 /// </summary>
00323         public Colour CodeBlockBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 245);
00324
00325 /// <summary>
00326 /// The colour for the bar to the left of block quotes.
00327 /// </summary>
00328         public Colour QuoteBlockBarColour { get; set; } = Colour.FromRgb(75, 152, 220);
00329
00330 /// <summary>
00331 /// The background colour for block quotes.
00332 /// </summary>
00333         public Colour QuoteBlockBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 255);
00334
00335 /// <summary>
00336 /// The colour for text that has been styled as "inserted".
00337 /// </summary>
00338         public Colour InsertedColour { get; set; } = Colour.FromRgb(0, 158, 115);
00339
00340 /// <summary>
00341 /// The colour for text that has been styled as "marked".
00342 /// </summary>
00343         public Colour MarkedColour { get; set; } = Colour.FromRgb(213, 94, 0);
00344
00345 /// <summary>
00346 /// The colour for the line separating the table header row from normal rows.
00347 /// </summary>
00348         public Colour TableHeaderRowSeparatorColour { get; set; } = Colours.Black;
00349
00350 /// <summary>
00351 /// The colour for lines separating table rows from each other.
00352 /// </summary>
00353         public Colour TableRowSeparatorColour { get; set; } = Colour.FromRgb(180, 180, 180);
00354
00355 /// <summary>
00356 /// The thickness of the line separating the table header row from normal rows.
00357 /// </summary>
00358         public double TableHeaderRowSeparatorThickness { get; set; } = 2;
00359
00360 /// <summary>
00361 /// The thickness of lines separating table rows from each other.
00362 /// </summary>
00363         public double TableHeaderSeparatorThickness { get; set; } = 1;
00364
00365 /// <summary>
00366 /// The bullet used for unchecked task list items.
00367 /// </summary>
00368         public Graphics TaskListUncheckedBullet { get; set; } = new Func<Graphics>(() =>
00369         {
00370             Graphics tbr = new Graphics();
00371
00372             GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3,
      -0.4).Arc(-0.3, -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0,
      Math.PI / 2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7,
      -0.2, 0.2, Math.PI, 3 * Math.PI / 2).Close();
00373             tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
00374             tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
00375
00376             return tbr;
00377         })();
00378
00379
00380 /// <summary>
00381 /// The bullet used for checked task list items.
00382 /// </summary>
00383         public Graphics TaskListCheckedBullet { get; set; } = new Func<Graphics>(() =>
00384         {
00385             Graphics tbr = new Graphics();
00386
00387             GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3,
      -0.4).Arc(-0.3, -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0,
      Math.PI / 2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7,
      -0.2, 0.2, Math.PI, 3 * Math.PI / 2).Close();
00388             tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
00389             tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
00390
00391             GraphicsPath tickpath = new GraphicsPath().MoveTo(-0.75, -0.1).LineTo(-0.5,
      0.15).LineTo(-0.1, -0.4);
00392
00393             tbr.StrokePath(new GraphicsPath().MoveTo(-0.5, 0.15).LineTo(-0.1, -0.4),
      Colour.FromRgb(240, 246, 249), 0.3, LineCaps.Round);
00394             tbr.StrokePath(tickpath, Colour.FromRgb(0, 158, 115), 0.2, LineCaps.Round);
00395
00396             return tbr;
00397         })();
```

```
00398
00399 /// <summary>
00400 /// Determines whether page breaks should be treated as such in the source.
00401 /// </summary>
00402         public bool AllowPageBreak { get; set; } = true;
00403
00404         internal MarkdownRenderer Clone()
00405         {
00406             return (MarkdownRenderer)this.MemberwiseClone();
00407         }
00408
00409         private MathPainter MathPainter = new MathPainter() { DisplayErrorInline = false };
00410
00411 /// <summary>
00412 /// Parses the supplied <paramref name="markdownSource"/> using all the supported extensions and
      renders the resulting document.  Page breaks are disabled, and the document is rendered as a single
      page with the specified <paramref name="width"/>.  The page will be cropped at the appropriate height
      to contain the entire document.
00413 /// </summary>
00414 /// <param name="markdownSource">The markdown source to parse.</param>
00415 /// <param name="width">The width of the page.</param>
00416 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Page"/> to a file.</param>
00417 /// <returns>A <see cref="Page"/> containing a rendering of the supplied markdown document.</returns>
00418         public Page RenderSinglePage(string markdownSource, double width, out Dictionary<string,
      string> linkDestinations)
00419         {
00420             MarkdownDocument document = Markdig.Markdown.Parse(markdownSource, new
      MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGenericAttributes().UseAutoIdentifiers
00421
00422             return this.RenderSinglePage(document, width, out linkDestinations);
00423         }
00424
00425 /// <summary>
00426 /// Renders the supplied <paramref name="markdownDocument"/>.  Page breaks are disabled, and the
      document is rendered as a single page with the specified <paramref name="width"/>.  The page will be
      cropped at the appropriate height to contain the entire document.
00427 /// </summary>
00428 /// <param name="markdownDocument">The markdown document to render.</param>
00429 /// <param name="width">The width of the page.</param>
00430 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Page"/> to a file.</param>
00431 /// <returns>A <see cref="Page"/> containing a rendering of the supplied markdown document.</returns>
00432         public Page RenderSinglePage(MarkdownDocument markdownDocument, double width, out
      Dictionary<string, string> linkDestinations)
00433         {
00434             Size prevPageSize = this.PageSize;
00435             bool allowPageBreak = this.AllowPageBreak;
00436
00437             this.PageSize = new Size(width, double.PositiveInfinity);
00438             this.AllowPageBreak = false;
00439
00440             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00441
00442             Graphics graphics = pag.Graphics;
00443
00444             graphics.Save();
00445
00446             graphics.Translate(Margins.Left, Margins.Top);
00447
00448             void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00449             {
00450
00451             }
00452
00453             MarkdownContext context = new MarkdownContext()
00454             {
00455                 Font = new Font(RegularFontFamily, BaseFontSize),
00456                 Cursor = new Point(0, 0),
00457                 Colour = ForegroundColour,
00458                 Underline = false,
00459                 Translation = new Point(this.Margins.Left, this.Margins.Top),
00460                 CurrentLine = null,
00461                 ListDepth = 0,
00462                 CurrentPage = pag
00463             };
00464
00465             int index = 0;
00466             foreach (Block block in markdownDocument)
00467             {
00468                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      markdownDocument.Count - 1);
00469                 index++;
00470             }
00471
```

```
00472             if (context.CurrentLine != null)
00473             {
00474                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00475             }
00476 
00477             graphics.Restore();
00478 
00479             pag.Crop(new Point(0, 0), new Size(width, context.BottomRight.Y + this.Margins.Bottom));
00480 
00481             linkDestinations = context.LinkDestinations;
00482             return pag;
00483         }
00484 
00485 /// <summary>
00486 /// Parses the supplied <paramref name="markdownSource"/> using all the supported extensions and
      renders the resulting document.  The <see cref="Document"/> produced consists of one or more pages of
      the size specified in the <see cref="PageSize"/> of the current instance.
00487 /// </summary>
00488 /// <param name="markdownSource">The markdown source to parse.</param>
00489 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Document"/> to a file.</param>
00490 /// <returns>A <see cref="Document"/> containing a rendering of the supplied markdown document,
      consisting of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.</returns>
00491         public Document Render(string markdownSource, out Dictionary<string, string> linkDestinations)
00492         {
00493             MarkdownDocument document = Markdig.Markdown.Parse(markdownSource, new
      MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGenericAttributes().UseAutoIdentifiers
00494 
00495             return this.Render(document, out linkDestinations);
00496         }
00497 
00498 /// <summary>
00499 /// Renders the supplied <paramref name="mardownDocument"/>.  The <see cref="Document"/> produced
      consists of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.
00500 /// </summary>
00501 /// <param name="mardownDocument">The markdown document to render.</param>
00502 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Document"/> to a file.</param>
00503 /// <returns>A <see cref="Document"/> containing a rendering of the supplied markdown document,
      consisting of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.</returns>
00504         public Document Render(MarkdownDocument mardownDocument, out Dictionary<string, string>
      linkDestinations)
00505         {
00506             Document doc = new Document();
00507             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00508             doc.Pages.Add(pag);
00509 
00510             Graphics graphics = pag.Graphics;
00511 
00512             graphics.Translate(Margins.Left, Margins.Top);
00513 
00514             void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00515             {
00516                 Page newPag = new Page(PageSize.Width, PageSize.Height) { Background =
      BackgroundColour };
00517                 doc.Pages.Add(newPag);
00518 
00519                 newPag.Graphics.Translate(mdContext.Translation);
00520                 mdContext.Cursor = new Point(0, 0);
00521                 mdContext.ForbiddenAreasLeft.Clear();
00522                 mdContext.ForbiddenAreasRight.Clear();
00523 
00524                 pageGraphics = newPag.Graphics;
00525                 mdContext.CurrentPage = newPag;
00526             }
00527 
00528             MarkdownContext context = new MarkdownContext()
00529             {
00530                 Font = new Font(RegularFontFamily, BaseFontSize),
00531                 Cursor = new Point(0, 0),
00532                 Colour = ForegroundColour,
00533                 Underline = false,
00534                 Translation = new Point(Margins.Left, Margins.Top),
00535                 CurrentLine = null,
00536                 ListDepth = 0,
00537                 CurrentPage = pag
00538             };
00539 
00540             int index = 0;
00541 
00542             foreach (Block block in mardownDocument)
```

```
00543             {
00544                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      mardownDocument.Count - 1);
00545                 index++;
00546             }
00547 
00548             if (context.CurrentLine != null)
00549             {
00550                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00551             }
00552 
00553             linkDestinations = context.LinkDestinations;
00554 
00555             return doc;
00556         }
00557 
00558         private Document RenderSubDocument(ContainerBlock document, ref MarkdownContext context)
00559         {
00560             Document doc = new Document();
00561             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00562             doc.Pages.Add(pag);
00563 
00564             Graphics graphics = pag.Graphics;
00565 
00566             graphics.Save();
00567 
00568             graphics.Translate(Margins.Left, Margins.Top);
00569 
00570             void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00571             {
00572                 pageGraphics.Restore();
00573                 Page newPag = new Page(PageSize.Width, PageSize.Height) { Background =
      BackgroundColour };
00574                 doc.Pages.Add(newPag);
00575 
00576                 newPag.Graphics.Save();
00577                 newPag.Graphics.Translate(mdContext.Translation);
00578                 mdContext.Cursor = new Point(0, 0);
00579                 mdContext.ForbiddenAreasLeft.Clear();
00580                 mdContext.ForbiddenAreasRight.Clear();
00581 
00582                 pageGraphics = newPag.Graphics;
00583                 mdContext.CurrentPage = newPag;
00584             }
00585 
00586             context.Translation = new Point(context.Translation.X + Margins.Left,
      context.Translation.Y + Margins.Top);
00587             context.CurrentPage = pag;
00588             context.ListDepth = 0;
00589 
00590             int index = 0;
00591             foreach (Block block in document)
00592             {
00593                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      document.Count - 1);
00594                 index++;
00595             }
00596 
00597             if (context.CurrentLine != null)
00598             {
00599                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00600             }
00601 
00602             graphics.Restore();
00603 
00604             return doc;
00605         }
00606 
00607         internal delegate void NewPageAction(ref MarkdownContext context, ref Graphics graphics);
00608 
00609         private void RenderBlock(Block block, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00610         {
00611             HtmlAttributes attributes = block.TryGetAttributes();
00612 
00613             if (attributes != null && !string.IsNullOrEmpty(attributes.Id))
00614             {
00615                 Point cursor = context.Cursor;
00616                 NewPageAction reversibleNewPageAction = (ref MarkdownContext currContext, ref Graphics
      currGraphics) =>
00617                 {
00618                     newPageAction(ref currContext, ref currGraphics);
00619                     cursor = currContext.Cursor;
00620                 };
00621 
```

```
00622                 RenderHTMLBlock("<a name=\"" + attributes.Id + "\"></a>", false, ref context, ref
      graphics, reversibleNewPageAction, spaceBefore, spaceAfter);
00623                 context.Cursor = cursor;
00624             }
00625 
00626             if (block is LeafBlock leaf)
00627             {
00628                 if (leaf is HeadingBlock heading)
00629                 {
00630                     RenderHeadingBlock(heading, ref context, ref graphics, newPageAction, spaceBefore,
      spaceAfter);
00631                 }
00632                 else if (leaf is ParagraphBlock paragraph)
00633                 {
00634                     RenderParagraphBlock(paragraph, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00635                 }
00636                 else if (leaf is CodeBlock code)
00637                 {
00638                     if (block is Markdig.Extensions.Mathematics.MathBlock math)
00639                     {
00640                         StringBuilder mathBuilder = new StringBuilder();
00641                         foreach (StringLine line in math.Lines)
00642                         {
00643                             mathBuilder.Append(line.ToString());
00644                             mathBuilder.Append("\n");
00645                         }
00646 
00647                         byte[] svgData;
00648 
00649                         using (MemoryStream ms = new MemoryStream())
00650                         {
00651                             MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Display;
00652                             MathPainter.FontSize = (float)(context.Font.FontSize *
      MathFontScalingFactor / ImageMultiplier);
00653                             MathPainter.LaTeX = mathBuilder.ToString();
00654                             Page pag = MathPainter.DrawToPage();
00655                             VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, ms);
00656                             svgData = ms.ToArray();
00657                         }
00658 
00659                         string imageUri = "<img align=\"center\" src=\"" +
      Convert.ToBase64String(svgData) + "\">";
00660                         RenderHTMLBlock(imageUri, false, ref context, ref graphics, newPageAction,
      true, true);
00661                     }
00662                     else if (leaf is FencedCodeBlock fenced)
00663                     {
00664                         if (!string.IsNullOrEmpty(fenced.Info))
00665                         {
00666                             RenderFencedCodeBlock(fenced, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00667                         }
00668                         else
00669                         {
00670                             RenderCodeBlock(code, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00671                         }
00672 
00673                     }
00674                     else
00675                     {
00676                         RenderCodeBlock(code, ref context, ref graphics, newPageAction, spaceBefore,
      spaceAfter);
00677                     }
00678                 }
00679                 else if (leaf is HtmlBlock html)
00680                 {
00681                     RenderHTMLBlock(html.Lines.ToString(), false, ref context, ref graphics,
      newPageAction, spaceBefore, spaceAfter);
00682                 }
00683                 else if (leaf is ThematicBreakBlock thematicBreak)
00684                 {
00685                     RenderThematicBreakBlock(thematicBreak, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00686                 }
00687                 else if (leaf is LinkReferenceDefinition link)
00688                 {
00689                     // Nothing to do (the links are correctly referenced by the parser)
00690                 }
00691 
00692             }
00693             else if (block is ContainerBlock)
00694             {
00695                 if (block is ListBlock list)
00696                 {
00697                     RenderListBlock(list, ref context, ref graphics, newPageAction);
```

```
00698                 }
00699                 else if (block is ListItemBlock listItem)
00700                 {
00701                     RenderListItemBlock(listItem, ref context, ref graphics, newPageAction);
00702                 }
00703                 else if (block is QuoteBlock quote)
00704                 {
00705                     RenderQuoteBlock(quote, ref context, ref graphics, newPageAction);
00706                 }
00707                 else if (block is LinkReferenceDefinitionGroup linkGroup)
00708                 {
00709                     // Nothing to render here
00710                 }
00711                 else if (block is Table table)
00712                 {
00713                     RenderTable(table, ref context, ref graphics, newPageAction);
00714                 }
00715             }
00716             else if (block is BlankLineBlock)
00717             {
00718                 // Nothing to render here
00719             }
00720         }
00721
00722         private void RenderHeadingBlock(HeadingBlock heading, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00723         {
00724             MarkdownContext prevContext = context.Clone();
00725
00726             context.Font = new Font(this.RegularFontFamily, BaseFontSize *
      HeaderFontSizeMultipliers[heading.Level - 1]);
00727
00728             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeHeading *
      context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeHeading *
      context.Font.FontSize - context.Font.Descent);
00729
00730             if (spaceBefore)
00731             {
00732                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeHeading * context.Font.FontSize);
00733             }
00734             else
00735             {
00736                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent);
00737             }
00738
00739             if (context.CurrentLine == null)
00740             {
00741                 context.CurrentLine = new Line(context.Font.Ascent);
00742             }
00743             else
00744             {
00745                 double delta = context.Cursor.Y - (prevContext.Cursor.Y + prevContext.Font.Ascent +
      SpaceBeforeParagaph * prevContext.Font.FontSize);
00746
00747                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
00748                 {
00749                     context.CurrentLine.Fragments[i].Translate(0, delta);
00750                 }
00751             }
00752
00753             foreach (Inline inline in heading.Inline)
00754             {
00755                 RenderInline(inline, ref context, ref graphics, newPageAction);
00756             }
00757
00758             if (this.HeaderLineThicknesses[heading.Level - 1] > 0)
00759             {
00760                 double lineY = context.Cursor.Y + context.Font.FontSize * 0.3;
00761                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(minX,
      context.Cursor.Y + context.Font.FontSize * 0.3), new Point(context.GetMaxX(lineY, PageSize.Width -
      Margins.Right - context.Translation.X), lineY), this.HeaderLineColour,
      this.HeaderLineThicknesses[heading.Level - 1], context.Tag));
00762             }
00763
00764             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00765             context.CurrentLine = null;
00766
00767             if (this.HeaderLineThicknesses[heading.Level - 1] > 0)
00768             {
00769                 double lineY = context.Cursor.Y + context.Font.FontSize * 0.3;
00770                 context.Cursor = new Point(context.Cursor.X, lineY +
      this.HeaderLineThicknesses[heading.Level - 1]);
00771             }
00772
00773             context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine *
```

```
context.Font.FontSize);
00774
00775             if (spaceAfter)
00776             {
00777                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterHeading *
context.Font.FontSize);
00778             }
00779
00780             prevContext.Cursor = context.Cursor;
00781             prevContext.BottomRight = context.BottomRight;
00782             prevContext.CurrentPage = context.CurrentPage;
00783             prevContext.CurrentLine = context.CurrentLine;
00784
00785             context = prevContext;
00786         }
00787
00788         private void RenderParagraphBlock(ParagraphBlock paragraph, ref MarkdownContext context, ref
Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00789         {
00790             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeParagaph *
context.Font.FontSize - context.Font.Descent);
00791
00792             if (spaceBefore)
00793             {
00794                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
SpaceBeforeParagaph * context.Font.FontSize);
00795             }
00796             else
00797             {
00798                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent);
00799             }
00800
00801             if (context.CurrentLine == null)
00802             {
00803                 context.CurrentLine = new Line(context.Font.Ascent);
00804             }
00805
00806             foreach (Inline inline in paragraph.Inline)
00807             {
00808                 RenderInline(inline, ref context, ref graphics, newPageAction);
00809             }
00810
00811             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
- this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00812             context.CurrentLine = null;
00813
00814             context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine *
context.Font.FontSize);
00815
00816             if (spaceAfter)
00817             {
00818                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
context.Font.FontSize);
00819             }
00820         }
00821
00822         private void RenderFencedCodeBlock(FencedCodeBlock codeBlock, ref MarkdownContext context, ref
Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00823         {
00824             string info = codeBlock.Info;
00825
00826             if (string.IsNullOrEmpty(info) || codeBlock.Lines.Count == 0)
00827             {
00828                 RenderCodeBlock(codeBlock, ref context, ref graphics, newPageAction, spaceBefore,
spaceAfter);
00829                 return;
00830             }
00831
00832             StringBuilder code = new StringBuilder();
00833
00834             foreach (StringLine line in codeBlock.Lines)
00835             {
00836                 code.Append(line.ToString());
00837                 code.Append('\n');
00838             }
00839
00840             List<List<FormattedString» lines = this.SyntaxHighlighter(code.ToString(0, code.Length -
1), info);
00841
00842             if (lines == null)
00843             {
00844                 RenderCodeBlock(codeBlock, ref context, ref graphics, newPageAction, spaceBefore,
spaceAfter);
00845                 return;
00846             }
00847
```

```
00848             MarkdownContext prevContext = context.Clone();
00849
00850             context.Font = new Font(this.CodeFont, context.Font.FontSize);
00851
00852             if (spaceBefore)
00853             {
00854                 context.Cursor = new Point(0, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize);
00855             }
00856
00857             int index = 0;
00858
00859             if (codeBlock.Lines.Count > 0)
00860             {
00861                 foreach (List<FormattedString> line in lines)
00862                 {
00863                     if (index < codeBlock.Lines.Count)
00864                     {
00865                         if (context.CurrentLine == null)
00866                         {
00867                             context.CurrentLine = new Line(context.Font.Ascent);
00868                         }
00869
00870                         double maxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00871
00872                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00873
00874                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y +
      context.Font.YMax);
00875
00876                         if (index == 0)
00877                         {
00878                             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new
      Point(minX, context.Cursor.Y - context.Font.YMax - this.SpaceAfterLine * context.Font.FontSize), new
      Size(maxX - minX, this.SpaceAfterLine * context.Font.FontSize * 2), CodeBlockBackgroundColour,
      context.Tag));
00879                         }
00880
00881                         foreach (FormattedString item in line)
00882                         {
00883                             context.Colour = item.Colour;
00884
00885                             if (!item.IsBold && !item.IsItalic)
00886                             {
00887                                 context.Font = new Font(this.CodeFont, context.Font.FontSize);
00888                             }
00889                             else if (item.IsBold && !item.IsItalic)
00890                             {
00891                                 context.Font = new Font(this.CodeFontBold, context.Font.FontSize);
00892                             }
00893                             else if (item.IsBold && item.IsItalic)
00894                             {
00895                                 context.Font = new Font(this.CodeFontBoldItalic,
      context.Font.FontSize);
00896                             }
00897                             else if (!item.IsBold && item.IsItalic)
00898                             {
00899                                 context.Font = new Font(this.CodeFontItalic, context.Font.FontSize);
00900                             }
00901
00902                             RenderCodeBlockLine(item.Text, ref context, ref graphics, newPageAction);
00903                         }
00904
00905
00906                         context.Colour = Colours.Black;
00907
00908                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00909                         context.CurrentLine = null;
00910
00911                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.YMin);
00912                         index++;
00913                     }
00914                     else
00915                     {
00916                         break;
00917                     }
00918                 }
00919             }
00920
00921             context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterLine * context.Font.FontSize);
00922
00923             if (spaceAfter)
00924             {
```

```
00925                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00926             }
00927
00928             prevContext.Cursor = context.Cursor;
00929             prevContext.BottomRight = context.BottomRight;
00930             prevContext.CurrentPage = context.CurrentPage;
00931             prevContext.CurrentLine = context.CurrentLine;
00932
00933             context = prevContext;
00934         }
00935
00936
00937         private void RenderCodeBlock(CodeBlock codeBlock, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00938         {
00939             MarkdownContext prevContext = context.Clone();
00940
00941             context.Font = new Font(this.CodeFont, context.Font.FontSize);
00942
00943             if (spaceBefore)
00944             {
00945                 context.Cursor = new Point(0, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize);
00946             }
00947
00948             int index = 0;
00949
00950             if (codeBlock.Lines.Count > 0)
00951             {
00952                 foreach (StringLine line in codeBlock.Lines)
00953                 {
00954                     if (index < codeBlock.Lines.Count)
00955                     {
00956                         if (context.CurrentLine == null)
00957                         {
00958                             context.CurrentLine = new Line(context.Font.Ascent);
00959                         }
00960
00961                         double maxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00962
00963                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00964
00965                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y +
      context.Font.YMax);
00966
00967                         if (index == 0)
00968                         {
00969                             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new
      Point(minX, context.Cursor.Y - context.Font.YMax - this.SpaceAfterLine * context.Font.FontSize), new
      Size(maxX - minX, this.SpaceAfterLine * context.Font.FontSize * 2), CodeBlockBackgroundColour,
      context.Tag));
00970                         }
00971
00972
00973                         RenderCodeBlockLine(line.ToString(), ref context, ref graphics,
      newPageAction);
00974
00975
00976                         context.Colour = Colours.Black;
00977
00978                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
00979                         context.CurrentLine = null;
00980
00981                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.YMin);
00982                         index++;
00983                     }
00984                     else
00985                     {
00986                         break;
00987                     }
00988                 }
00989             }
00990
00991             context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterLine * context.Font.FontSize);
00992
00993             if (spaceAfter)
00994             {
00995                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00996             }
00997
00998             prevContext.Cursor = context.Cursor;
```

```
00999             prevContext.BottomRight = context.BottomRight;
01000             prevContext.CurrentPage = context.CurrentPage;
01001             prevContext.CurrentLine = context.CurrentLine;
01002 
01003             context = prevContext;
01004         }
01005 
01006         private void RenderThematicBreakBlock(ThematicBreakBlock thematicBreak, ref MarkdownContext
      context, ref Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
01007         {
01008             if (context.CurrentLine == null)
01009             {
01010                 context.CurrentLine = new Line(0);
01011             }
01012 
01013             if (spaceBefore)
01014             {
01015                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize + this.ThematicBreakThickness * 0.5);
01016             }
01017             else
01018             {
01019                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      this.ThematicBreakThickness * 0.5);
01020             }
01021 
01022 
01023             double maxX = context.GetMaxX(context.Cursor.Y, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
01024             double minX = context.GetMinX(context.Cursor.Y);
01025 
01026             context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(minX, context.Cursor.Y),
      new Point(maxX, context.Cursor.Y), this.ThematicBreakLineColour, this.ThematicBreakThickness,
      context.Tag));
01027 
01028             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01029             context.CurrentLine = null;
01030 
01031             if (spaceAfter)
01032             {
01033                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize + this.ThematicBreakThickness * 0.5);
01034             }
01035             else
01036             {
01037                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      this.ThematicBreakThickness * 0.5);
01038             }
01039         }
01040 
01041         private void RenderInline(Inline inline, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction)
01042         {
01043             HtmlAttributes attributes = inline.TryGetAttributes();
01044 
01045             if (attributes != null && !string.IsNullOrEmpty(attributes.Id))
01046             {
01047                 Point cursor = context.Cursor;
01048                 RenderHTMLBlock("<a name=\"" + attributes.Id + "\"></a>", true, ref context, ref
      graphics, newPageAction, false, false);
01049                 context.Cursor = cursor;
01050             }
01051 
01052             if (inline is LeafInline)
01053             {
01054                 if (inline is AutolinkInline autoLink)
01055                 {
01056                     LinkInline link = new LinkInline((autoLink.IsEmail ?  "mailto:" :  "") +
      autoLink.Url, "");
01057                     link.AppendChild(new LiteralInline(autoLink.Url));
01058 
01059                     RenderLinkInline(link, ref context, ref graphics, newPageAction);
01060                 }
01061                 else if (inline is CodeInline code)
01062                 {
01063                     RenderCodeInline(code, ref context, ref graphics, newPageAction);
01064                 }
01065                 else if (inline is HtmlEntityInline htmlEntity)
01066                 {
01067                     RenderLiteralInline(new LiteralInline(htmlEntity.Transcoded), ref context, ref
      graphics, newPageAction);
01068                 }
01069                 else if (inline is HtmlInline html)
01070                 {
01071                     RenderHTMLBlock(html.Tag, true, ref context, ref graphics, newPageAction, true,
      true);
```

```
01072                 }
01073                 else if (inline is LineBreakInline lineBreak)
01074                 {
01075                     RenderLineBreakInline(lineBreak.IsHard, false, ref context, ref graphics,
      newPageAction);
01076                 }
01077                 else if (inline is LiteralInline literal)
01078                 {
01079                     RenderLiteralInline(literal, ref context, ref graphics, newPageAction);
01080                 }
01081                 else if (inline is Markdig.Extensions.Mathematics.MathInline math)
01082                 {
01083                     byte[] svgData;
01084 
01085                     using (MemoryStream ms = new MemoryStream())
01086                     {
01087                         if (math.DelimiterCount == 1)
01088                         {
01089                             MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Text;
01090                         }
01091                         else
01092                         {
01093                             MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Display;
01094                         }
01095 
01096                         MathPainter.FontSize = (float)(context.Font.FontSize * MathFontScalingFactor /
      ImageMultiplier);
01097 
01098                         MathPainter.LaTeX = math.Content.ToString();
01099                         Page pag = MathPainter.DrawToPage();
01100                         VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, ms);
01101                         svgData = ms.ToArray();
01102                     }
01103 
01104                     string imageUri = "<img src=\"" +
      Convert.ToBase64String(svgData) + "\">";
01105                     RenderHTMLBlock(imageUri, true, ref context, ref graphics, newPageAction, true,
      true);
01106                 }
01107                 else if (inline is Markdig.Extensions.SmartyPants.SmartyPant smartyPant)
01108                 {
01109                     switch (smartyPant.Type)
01110                     {
01111                         case Markdig.Extensions.SmartyPants.SmartyPantType.LeftDoubleQuote:
01112                             RenderLiteralInline(new LiteralInline("“"), ref context, ref graphics,
      newPageAction);
01113                             break;
01114                         case Markdig.Extensions.SmartyPants.SmartyPantType.RightDoubleQuote:
01115                             RenderLiteralInline(new LiteralInline("”"), ref context, ref graphics,
      newPageAction);
01116                             break;
01117                         case Markdig.Extensions.SmartyPants.SmartyPantType.LeftQuote:
01118                             RenderLiteralInline(new LiteralInline("‘"), ref context, ref graphics,
      newPageAction);
01119                             break;
01120                         case Markdig.Extensions.SmartyPants.SmartyPantType.RightQuote:
01121                             RenderLiteralInline(new LiteralInline("’"), ref context, ref graphics,
      newPageAction);
01122                             break;
01123                         case Markdig.Extensions.SmartyPants.SmartyPantType.Dash2:
01124                             RenderLiteralInline(new LiteralInline("–"), ref context, ref graphics,
      newPageAction);
01125                             break;
01126                         case Markdig.Extensions.SmartyPants.SmartyPantType.Dash3:
01127                             RenderLiteralInline(new LiteralInline("—"), ref context, ref graphics,
      newPageAction);
01128                             break;
01129                         case Markdig.Extensions.SmartyPants.SmartyPantType.DoubleQuote:
01130                             RenderLiteralInline(new LiteralInline("\""), ref context, ref graphics,
      newPageAction);
01131                             break;
01132                         case Markdig.Extensions.SmartyPants.SmartyPantType.Ellipsis:
01133                             RenderLiteralInline(new LiteralInline("..."), ref context, ref graphics,
      newPageAction);
01134                             break;
01135                         case Markdig.Extensions.SmartyPants.SmartyPantType.LeftAngleQuote:
01136                             RenderLiteralInline(new LiteralInline("«"), ref context, ref graphics,
      newPageAction);
01137                             break;
01138                         case Markdig.Extensions.SmartyPants.SmartyPantType.Quote:
01139                             RenderLiteralInline(new LiteralInline("’"), ref context, ref graphics,
      newPageAction);
01140                             break;
01141                         case Markdig.Extensions.SmartyPants.SmartyPantType.RightAngleQuote:
01142                             RenderLiteralInline(new LiteralInline("»"), ref context, ref graphics,
      newPageAction);
01143                             break;
```

```
01144                     }
01145                 }
01146                 else if (inline is Markdig.Extensions.TaskLists.TaskList)
01147                 {
01148                     // Nothing to render here (the checkbox has already been rendered)
01149                 }
01150             }
01151             else if (inline is ContainerInline)
01152             {
01153                 if (inline is DelimiterInline)
01154                 {
01155                     // Nothing to render here
01156                 }
01157                 else if (inline is EmphasisInline emphasis)
01158                 {
01159                     RenderEmphasisInline(emphasis, ref context, ref graphics, newPageAction);
01160                 }
01161                 else if (inline is LinkInline link)
01162                 {
01163                     RenderLinkInline(link, ref context, ref graphics, newPageAction);
01164                 }
01165             }
01166         }
01167 
01168         private void RenderLineBreakInline(bool isHard, bool isPageBreak, ref MarkdownContext context,
    ref Graphics graphics, NewPageAction newPageAction)
01169         {
01170             if (isHard)
01171             {
01172                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
    this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01173                 context.CurrentLine = new Line(context.Font.Ascent);
01174                 context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine
    * context.Font.FontSize + context.Font.Ascent);
01175 
01176                 double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y
    - context.Font.Descent);
01177 
01178                 context.Cursor = new Point(minX, context.Cursor.Y);
01179 
01180                 if (isPageBreak)
01181                 {
01182                     newPageAction(ref context, ref graphics);
01183                 }
01184             }
01185             else
01186             {
01187                 double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') /
    1000.0 * context.Font.FontSize;
01188                 context.Cursor = new Point(context.Cursor.X + spaceWidth, context.Cursor.Y);
01189             }
01190         }
01191 
01192         private void RenderLiteralInline(LiteralInline literal, ref MarkdownContext context, ref
    Graphics graphics, NewPageAction newPageAction)
01193         {
01194             string text = literal.Content.ToString();
01195 
01196             double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
    * context.Font.FontSize;
01197 
01198             List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
    this.Margins.Right - context.Translation.X - context.MarginBottomRight.X).ToList();
01199 
01200             double underlineStart = context.Cursor.X;
01201             double underlineEnd = context.Cursor.X;
01202 
01203             double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
    context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
    context.Translation.X - context.MarginBottomRight.X);
01204 
01205             bool ignoreNextWhitespace = false;
01206 
01207             bool broken = false;
01208 
01209             for (int i = 0; i < words.Count; i++)
01210             {
01211                 Word w = words[i];
01212 
01213                 if (!string.IsNullOrEmpty(w.Text))
01214                 {
01215                     if (!ignoreNextWhitespace)
01216                     {
01217                         context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
    (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
01218                     }
01219                     else
```

```
01220                     {
01221                         ignoreNextWhitespace = false;
01222                     }
01223 
01224                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
01225 
01226                     double finalX = context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
01227 
01228                     if (finalX <= currLineMaxX || broken)
01229                     {
01230                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      + wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
01231                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
01232 
01233                         broken = false;
01234 
01235                         if (context.Underline || context.StrikeThrough)
01236                         {
01237                             underlineEnd = context.Cursor.X;
01238                         }
01239                     }
01240                     else
01241                     {
01242                         if (context.Underline && underlineStart != underlineEnd)
01243                         {
01244                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
      context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
      (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :  this.UnderlineThickness),
      context.Tag));
01245                         }
01246                         else if (context.StrikeThrough && underlineStart != underlineEnd)
01247                         {
01248                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
      Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
      context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness
      :  this.UnderlineThickness), context.Tag));
01249                         }
01250 
01251                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01252 
01253 
01254 
01255                         context.CurrentLine = new Line(context.Font.Ascent);
01256                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
      SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
01257                         currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
01258 
01259                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
01260 
01261                         context.Cursor = new Point(minX, context.Cursor.Y);
01262 
01263                         underlineStart = minX;
01264                         underlineEnd = minX;
01265 
01266                         i--;
01267                         ignoreNextWhitespace = true;
01268                         broken = true;
01269                     }
01270                 }
01271                 else
01272                 {
01273                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
01274                 }
01275             }
01276 
01277             if (context.Underline && underlineStart != underlineEnd)
01278             {
01279                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
      context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
      ?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
01280             }
01281             else if (context.StrikeThrough && underlineStart != underlineEnd)
01282             {
01283                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
      context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
```

```
this.UnderlineThickness), context.Tag));
01284             }
01285         }
01286 
01287         private void RenderCodeInline(CodeInline code, ref MarkdownContext context, ref Graphics
graphics, NewPageAction newPageAction)
01288         {
01289             MarkdownContext prevContext = context.Clone();
01290 
01291             context.Font = new Font(this.CodeFont, context.Font.FontSize);
01292 
01293             double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
* context.Font.FontSize;
01294 
01295             string text = code.Content.ToString();
01296             List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
this.Margins.Right - context.Translation.X - context.MarginBottomRight.X).ToList();
01297 
01298             double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
context.Translation.X - context.MarginBottomRight.X);
01299 
01300             context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
context.Font.FontSize, context.Cursor.Y);
01301 
01302 
01303             double underlineStart = context.Cursor.X;
01304             double underlineEnd = context.Cursor.X;
01305 
01306             bool broken = false;
01307 
01308             for (int i = 0; i < words.Count; i++)
01309             {
01310                 Word w = words[i];
01311 
01312                 if (!string.IsNullOrEmpty(w.Text))
01313                 {
01314                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
(w.PrecedingWhitespace == '\t' ? 4 : 1), context.Cursor.Y);
01315 
01316                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
01317 
01318                     double finalX = context.Cursor.X + wordMetrics.Width +
wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
01319 
01320                     if (finalX <= currLineMaxX || broken)
01321                     {
01322                         if (i == 0)
01323                         {
01324                             context.CurrentLine.Fragments.Add(new RectangleFragment(new
Point(context.Cursor.X - this.CodeInlineMargin * context.Font.FontSize, context.Cursor.Y -
context.Font.YMax), new Size(this.CodeInlineMargin * context.Font.FontSize, context.Font.YMax -
context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
01325                         }
01326 
01327                         context.CurrentLine.Fragments.Add(new RectangleFragment(new
Point(context.Cursor.X - spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ? 4 : 1),
context.Cursor.Y - context.Font.YMax), new Size(wordMetrics.Width + wordMetrics.LeftSideBearing * 2 +
wordMetrics.RightSideBearing + spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ? 4 :
1), context.Font.YMax - context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
01328 
01329                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
+ wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
01330                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
01331 
01332                         broken = false;
01333 
01334                         if (context.Underline || context.StrikeThrough)
01335                         {
01336                             underlineEnd = context.Cursor.X;
01337                         }
01338                     }
01339                     else
01340                     {
01341                         if (context.Underline && underlineStart != underlineEnd)
01342                         {
01343                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
(context.Font.FontFamily.IsBold ? this.BoldUnderlineThickness : this.UnderlineThickness),
context.Tag));
01344                         }
01345                         else if (context.StrikeThrough && underlineStart != underlineEnd)
01346                         {
01347                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
```

```
Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness
:  this.UnderlineThickness), context.Tag));
01348                       }
01349
01350                       context.CurrentLine.Render(ref graphics, ref context, newPageAction,
this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01351
01352                       context.CurrentLine = new Line(prevContext.Font.Ascent);
01353
01354                       context.Cursor = new Point(0, context.Cursor.Y - prevContext.Font.Descent +
SpaceAfterLine * prevContext.Font.FontSize + prevContext.Font.Ascent);
01355
01356
01357                       double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
context.Cursor.Y - context.Font.Descent);
01358
01359                       context.Cursor = new Point(minX, context.Cursor.Y);
01360
01361                       if (i == 0)
01362                       {
01363                           context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
context.Font.FontSize, context.Cursor.Y);
01364                       }
01365
01366                       underlineStart = minX;
01367                       underlineEnd = minX;
01368
01369                       i--;
01370                       broken = true;
01371                   }
01372               }
01373           }
01374
01375
01376           if (context.Underline && underlineStart != underlineEnd)
01377           {
01378               context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
01379           }
01380           else if (context.StrikeThrough && underlineStart != underlineEnd)
01381           {
01382               context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
this.UnderlineThickness), context.Tag));
01383           }
01384
01385           context.CurrentLine.Fragments.Add(new RectangleFragment(new Point(context.Cursor.X,
context.Cursor.Y - context.Font.YMax), new Size(this.CodeInlineMargin * context.Font.FontSize,
context.Font.YMax - context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
01386
01387           context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
context.Font.FontSize, context.Cursor.Y);
01388
01389           prevContext.Cursor = context.Cursor;
01390           prevContext.BottomRight = context.BottomRight;
01391           prevContext.CurrentPage = context.CurrentPage;
01392           prevContext.CurrentLine = context.CurrentLine;
01393
01394           context = prevContext;
01395       }
01396
01397       private void RenderCodeBlockLine(string text, ref MarkdownContext context, ref Graphics
graphics, NewPageAction newPageAction)
01398       {
01399           double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
* context.Font.FontSize;
01400
01401           List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
this.Margins.Right - context.Translation.X - context.Font.FontSize * 2 -
context.MarginBottomRight.X).ToList();
01402
01403           double underlineStart = context.Cursor.X;
01404           double underlineEnd = context.Cursor.X;
01405
01406           double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y -
context.Font.Descent);
01407
01408           double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
context.Translation.X - context.MarginBottomRight.X) - context.Font.FontSize;
01409
01410           bool broken = false;
```

```
01411
01412             for (int i = 0; i < words.Count; i++)
01413             {
01414                 Word w = words[i];
01415                 if (!string.IsNullOrEmpty(w.Text))
01416                 {
01417                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ? 4 : 1), context.Cursor.Y);
01418                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
01419
01420                     double finalX = context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
01421
01422                     double effW = wordMetrics.Width + wordMetrics.RightSideBearing +
      wordMetrics.LeftSideBearing;
01423
01424                     double maxW = this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.Font.FontSize * 2 - spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ? 4 : 1)
      - context.MarginBottomRight.X;
01425
01426                     if (finalX <= currLineMaxX || broken)
01427                     {
01428                         broken = false;
01429                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      + wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
01430                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
01431
01432                         if (context.Underline || context.StrikeThrough)
01433                         {
01434                             underlineEnd = context.Cursor.X;
01435                         }
01436                     }
01437                     else
01438                     {
01439                         context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new Point(minX,
      context.Cursor.Y - context.Font.YMax), new Size(currLineMaxX + context.Font.FontSize - minX,
      context.Font.YMax - context.Font.YMin + this.SpaceAfterLine * context.Font.FontSize),
      CodeBlockBackgroundColour, context.Tag));
01440
01441                         if (context.Underline && underlineStart != underlineEnd)
01442                         {
01443                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
      context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
      (context.Font.FontFamily.IsBold ? this.BoldUnderlineThickness : this.UnderlineThickness),
      context.Tag));
01444                         }
01445                         else if (context.StrikeThrough && underlineStart != underlineEnd)
01446                         {
01447                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
      Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
      context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ? this.BoldUnderlineThickness
      : this.UnderlineThickness), context.Tag));
01448                         }
01449
01450                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01451
01452                         underlineStart = 0;
01453                         underlineEnd = 0;
01454
01455                         context.CurrentLine = new Line(context.Font.Ascent);
01456
01457                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
      SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
01458
01459                         currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X) - context.Font.FontSize;
01460
01461                         minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
01462
01463                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y);
01464
01465                         i--;
01466                         broken = true;
01467                     }
01468                 }
01469                 else
01470                 {
01471                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ? 4 : 1), context.Cursor.Y);
01472                 }
01473             }
```

```
01474
01475             if (context.Underline && underlineStart != underlineEnd)
01476             {
01477                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
      context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
      ?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
01478             }
01479             else if (context.StrikeThrough && underlineStart != underlineEnd)
01480             {
01481                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
      context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
      this.UnderlineThickness), context.Tag));
01482             }
01483
01484             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new Point(minX,
      context.Cursor.Y - context.Font.YMax), new Size(currLineMaxX + context.Font.FontSize - minX,
      context.Font.YMax - context.Font.YMin + this.SpaceAfterLine * context.Font.FontSize),
      CodeBlockBackgroundColour, context.Tag));
01485         }
01486
01487         private void RenderEmphasisInline(EmphasisInline emphasis, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction)
01488         {
01489             MarkdownContext prevContext = context.Clone();
01490
01491             Point translationToUndo = new Point(0, 0);
01492
01493             switch (emphasis.DelimiterChar)
01494             {
01495                 case '*':
01496                 case '_':
01497                     if (emphasis.DelimiterCount == 2)
01498                     {
01499                         if (context.Font.FontFamily == this.ItalicFontFamily)
01500                         {
01501                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
01502                         }
01503                         else if (context.Font.FontFamily == this.BoldFontFamily)
01504                         {
01505                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
01506                         }
01507                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
01508                         {
01509                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
01510                         }
01511                         else
01512                         {
01513                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
01514                         }
01515                     }
01516                     else if (emphasis.DelimiterCount == 3)
01517                     {
01518                         if (context.Font.FontFamily == this.ItalicFontFamily)
01519                         {
01520                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
01521                         }
01522                         else if (context.Font.FontFamily == this.BoldFontFamily)
01523                         {
01524                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
01525                         }
01526                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
01527                         {
01528                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
01529                         }
01530                         else
01531                         {
01532                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
01533                         }
01534                     }
01535                     else
01536                     {
01537                         if (context.Font.FontFamily == this.ItalicFontFamily)
01538                         {
01539                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
01540                         }
01541                         else if (context.Font.FontFamily == this.BoldFontFamily)
01542                         {
01543                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
01544                         }
01545                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
01546                         {
01547                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
01548                         }
01549                         else
```

```
01550                         {
01551                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
01552                         }
01553                     }
01554                     break;
01555                 case '"':
01556                     if (emphasis.DelimiterCount == 2)
01557                     {
01558                         if (context.Font.FontFamily == this.ItalicFontFamily)
01559                         {
01560                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
01561                         }
01562                         else if (context.Font.FontFamily == this.BoldFontFamily)
01563                         {
01564                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
01565                         }
01566                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
01567                         {
01568                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
01569                         }
01570                         else
01571                         {
01572                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
01573                         }
01574                     }
01575                     break;
01576                 case '~':
01577                     if (emphasis.DelimiterCount == 1)
01578                     {
01579                         //subscript;
01580                         context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
    context.Font.FontSize * this.SubscriptShift);
01581                         translationToUndo = new Point(translationToUndo.X, translationToUndo.Y +
    context.Font.FontSize * this.SubscriptShift);
01582                         context.Font = new Font(context.Font.FontFamily, context.Font.FontSize *
    this.SubSuperscriptFontSize);
01583                     }
01584                     else
01585                     {
01586                         //strikethrough
01587                         context.StrikeThrough = true;
01588                     }
01589                     break;
01590                 case '^':
01591                     if (emphasis.DelimiterCount == 1)
01592                     {
01593                         //superscript
01594                         context.Cursor = new Point(context.Cursor.X, context.Cursor.Y -
    context.Font.FontSize * this.SuperscriptShift);
01595                         translationToUndo = new Point(translationToUndo.X, translationToUndo.Y -
    context.Font.FontSize * this.SuperscriptShift);
01596                         context.Font = new Font(context.Font.FontFamily, context.Font.FontSize *
    this.SubSuperscriptFontSize);
01597                     }
01598                     break;
01599                 case '+':
01600                     context.Colour = this.InsertedColour;
01601                     break;
01602                 case '=':
01603                     context.Colour = this.MarkedColour;
01604                     break;
01605             }
01606
01607             foreach (Inline innerInline in emphasis)
01608             {
01609                 RenderInline(innerInline, ref context, ref graphics, newPageAction);
01610             }
01611
01612             if (translationToUndo.X != 0 || translationToUndo.Y != 0)
01613             {
01614                 context.Cursor = new Point(context.Cursor.X - translationToUndo.X, context.Cursor.Y -
    translationToUndo.Y);
01615             }
01616
01617
01618             prevContext.Cursor = context.Cursor;
01619             prevContext.BottomRight = context.BottomRight;
01620             prevContext.CurrentPage = context.CurrentPage;
01621             prevContext.CurrentLine = context.CurrentLine;
01622
01623             context = prevContext;
01624         }
01625
01626         private void RenderLinkInline(LinkInline link, ref MarkdownContext context, ref Graphics
    graphics, NewPageAction newPageAction)
01627         {
01628             if (!link.IsImage)
```

```
01629             {
01630                 MarkdownContext prevContext = context.Clone();
01631 
01632                 context.Colour = this.LinkColour;
01633                 context.Underline = true;
01634                 string tag = Guid.NewGuid().ToString("N");
01635 
01636                 if (!link.Url.StartsWith("#"))
01637                 {
01638                     if (Uri.TryCreate(this.BaseLinkUri, link.Url, out Uri uri))
01639                     {
01640                         context.LinkDestinations[tag] = this.LinkUriResolver(uri.ToString());
01641                     }
01642                     else
01643                     {
01644                         context.LinkDestinations[tag] = this.LinkUriResolver(link.Url);
01645                     }
01646                 }
01647                 else
01648                 {
01649                     if (!context.InternalAnchors.TryGetValue(link.Url, out string anchor))
01650                     {
01651                         anchor = Guid.NewGuid().ToString("N");
01652                         context.InternalAnchors[link.Url] = anchor;
01653                     }
01654 
01655                     context.LinkDestinations[tag] = "#" + anchor;
01656                 }
01657 
01658                 context.Tag = tag;
01659 
01660                 foreach (Inline innerInline in link)
01661                 {
01662                     RenderInline(innerInline, ref context, ref graphics, newPageAction);
01663                 }
01664 
01665                 prevContext.Cursor = context.Cursor;
01666                 prevContext.BottomRight = context.BottomRight;
01667                 prevContext.CurrentPage = context.CurrentPage;
01668                 prevContext.CurrentLine = context.CurrentLine;
01669 
01670                 context = prevContext;
01671             }
01672             else
01673             {
01674                 HtmlTag tag = HtmlTag.Parse("<img src=\"" + link.Url.Replace("\"", "\\\"") +
      "\">").FirstOrDefault();
01675 
01676                 RenderHTMLImage(tag, true, ref context, ref graphics, newPageAction);
01677             }
01678         }
01679 
01680         private void RenderListBlock(ListBlock list, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction)
01681         {
01682             MarkdownContext prevContext = context.Clone();
01683 
01684             context.ListDepth++;
01685 
01686             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize, context.Cursor.Y - context.Font.Descent + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
01687 
01688             if (context.CurrentLine != null)
01689             {
01690                 foreach (LineFragment fragment in context.CurrentLine.Fragments)
01691                 {
01692                     fragment.Translate(-minX - this.IndentWidth, 0);
01693                 }
01694             }
01695 
01696             graphics.Translate(minX + this.IndentWidth, 0);
01697             context.Translation = new Point(context.Translation.X + minX + this.IndentWidth,
      context.Translation.Y);
01698 
01699             foreach (Block block in list)
01700             {
01701                 RenderBlock(block, ref context, ref graphics, newPageAction, true, true);
01702             }
01703 
01704             graphics.Translate(-minX - this.IndentWidth, 0);
01705             context.Translation = new Point(context.Translation.X - minX - this.IndentWidth,
      context.Translation.Y);
01706 
01707             prevContext.Cursor = context.Cursor;
01708             prevContext.BottomRight = context.BottomRight;
01709             prevContext.CurrentPage = context.CurrentPage;
```

```
01710             prevContext.CurrentLine = context.CurrentLine;
01711
01712             context = prevContext;
01713         }
01714
01715         private void RenderQuoteBlock(QuoteBlock quote, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction)
01716         {
01717             MarkdownContext prevContext = context.Clone();
01718
01719             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize, context.Cursor.Y - context.Font.Descent + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
01720
01721             Graphics quoteGraphics = new Graphics();
01722
01723             quoteGraphics.Translate(minX + this.QuoteBlockIndentWidth, 0);
01724             context.Translation = new Point(minX + this.QuoteBlockIndentWidth, 0);
01725             context.MarginBottomRight = new Point(context.MarginBottomRight.X +
      prevContext.Translation.X, context.MarginBottomRight.Y + prevContext.Translation.Y);
01726
01727             double maxX = this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X;
01728
01729             double startY = context.Cursor.Y + context.Font.Ascent - context.Font.YMax;
01730
01731             Point currTranslation = context.Translation;
01732
01733             Graphics parentGraphics = graphics;
01734
01735             NewPageAction newPageActionWithBlockquotes = (ref MarkdownContext currContext, ref
      Graphics currGraphics) =>
01736             {
01737                 double currEndY = currContext.Cursor.Y;
01738
01739                 double currMaxX = maxX;
01740
01741                 Graphics currBackgroundGraphics = new Graphics();
01742
01743                 currBackgroundGraphics.Save();
01744
01745                 currBackgroundGraphics.Translate(currContext.Translation);
01746                 currBackgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth +
      this.QuoteBlockBarWidth, startY), new Size(currMaxX + this.QuoteBlockIndentWidth -
      this.QuoteBlockBarWidth, currEndY - startY), this.QuoteBlockBackgroundColour, tag:  currContext.Tag);
01747                 currBackgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth, startY),
      new Size(this.QuoteBlockBarWidth, currEndY - startY), this.QuoteBlockBarColour, tag:
      currContext.Tag);
01748
01749                 currBackgroundGraphics.Restore();
01750
01751                 currBackgroundGraphics.DrawGraphics(0, 0, currGraphics);
01752
01753                 parentGraphics.DrawGraphics(0, 0, currBackgroundGraphics);
01754
01755                 Point currContextTranslation = currContext.Translation;
01756
01757                 currContext.Translation = prevContext.Translation;
01758
01759                 newPageAction(ref currContext, ref parentGraphics);
01760
01761                 currContext.Translation = currContextTranslation;
01762
01763                 currGraphics = new Graphics();
01764                 currGraphics.Translate(minX + this.QuoteBlockIndentWidth, 0);
01765
01766                 startY = currContext.Cursor.Y + currContext.Font.Ascent - currContext.Font.YMax;
01767             };
01768
01769             int index = 0;
01770
01771             foreach (Block block in quote)
01772             {
01773                 RenderBlock(block, ref context, ref quoteGraphics, newPageActionWithBlockquotes, true,
      index < quote.Count - 1);
01774                 index++;
01775             }
01776
01777             double endY = context.Cursor.Y;
01778
01779             Graphics backgroundGraphics = new Graphics();
01780
01781             backgroundGraphics.Save();
01782
01783             backgroundGraphics.Translate(context.Translation);
01784             backgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth +
      this.QuoteBlockBarWidth, startY), new Size(maxX + this.QuoteBlockIndentWidth -
```

```
    this.QuoteBlockBarWidth, endY - startY), this.QuoteBlockBackgroundColour, tag:  context.Tag);
01785            backgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth, startY), new
    Size(this.QuoteBlockBarWidth, endY - startY), this.QuoteBlockBarColour, tag:  context.Tag);
01786
01787            backgroundGraphics.Restore();
01788
01789            backgroundGraphics.DrawGraphics(0, 0, quoteGraphics);
01790
01791            parentGraphics.DrawGraphics(0, 0, backgroundGraphics);
01792
01793            graphics = parentGraphics;
01794
01795            context.Translation = prevContext.Translation;
01796
01797            if (!(quote.Parent is QuoteBlock) || quote.Parent.LastChild != quote)
01798            {
01799                context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceAfterParagraph *
    context.Font.FontSize);
01800            }
01801
01802            prevContext.Cursor = context.Cursor;
01803            prevContext.BottomRight = context.BottomRight;
01804            prevContext.CurrentPage = context.CurrentPage;
01805            prevContext.CurrentLine = context.CurrentLine;
01806
01807            context = prevContext;
01808        }
01809
01810        private void RenderListItemBlock(ListItemBlock listItem, ref MarkdownContext context, ref
    Graphics graphics, NewPageAction newPageAction)
01811        {
01812            MarkdownContext prevContext = context.Clone();
01813
01814            bool isLoose = false;
01815
01816            double startX = context.Cursor.X;
01817            double startY = context.Cursor.Y;
01818
01819            if (listItem.Parent is ListBlock list)
01820            {
01821                if (list.IsOrdered)
01822                {
01823                    if (context.CurrentLine == null)
01824                    {
01825                        context.CurrentLine = new Line(context.Font.Ascent);
01826                    }
01827
01828                    string bullet;
01829
01830                    switch (list.BulletType)
01831                    {
01832                        case 'a':
01833                        case 'A':
01834                            bullet = ((char)((int)list.DefaultOrderedStart[0] + listItem.Order -
    1)).ToString() + list.OrderedDelimiter;
01835                            break;
01836                        case 'i':
01837                            bullet = GetRomanNumeral(listItem.Order).ToLower() +
    list.OrderedDelimiter;
01838                            break;
01839                        case 'I':
01840                            bullet = GetRomanNumeral(listItem.Order) + list.OrderedDelimiter;
01841                            break;
01842                        default:
01843                            bullet = listItem.Order.ToString() + list.OrderedDelimiter;
01844                            break;
01845                    }
01846
01847                    if (list.IsLoose)
01848                    {
01849                        isLoose = true;
01850                        context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
    - context.Font.MeasureText(bullet).Width - this.IndentWidth * 0.15, context.Cursor.Y +
    context.Font.Ascent + SpaceBeforeParagaph * context.Font.FontSize), bullet, context.Font,
    context.Colour, context.Tag));
01851                    }
01852                    else
01853                    {
01854                        isLoose = false;
01855                        context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
    - context.Font.MeasureText(bullet).Width - this.IndentWidth * 0.15, context.Cursor.Y +
    context.Font.Ascent), bullet, context.Font, context.Colour, context.Tag));
01856                    }
01857                }
01858                else
01859                {
01860                    if (listItem.Count > 0 && listItem[0] is ParagraphBlock paragraph &&
```

```
    paragraph.Inline?.FirstChild is Markdig.Extensions.TaskLists.TaskList task)
01861                   {
01862                       if (context.CurrentLine == null)
01863                       {
01864                           context.CurrentLine = new Line(context.Font.Ascent);
01865                       }
01866
01867                       Graphics bullet = new Graphics();
01868                       bullet.Scale(context.Font.FontSize, context.Font.FontSize);
01869
01870                       if (task.Checked)
01871                       {
01872                           bullet.DrawGraphics(0, 0, this.TaskListCheckedBullet);
01873                       }
01874                       else
01875                       {
01876                           bullet.DrawGraphics(0, 0, this.TaskListUncheckedBullet);
01877                       }
01878
01879                       if (list.IsLoose)
01880                       {
01881                           isLoose = true;
01882                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
    Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent +
    SpaceBeforeParagaph * context.Font.FontSize - context.Font.Descent * 0.5 - (context.Font.Ascent -
    context.Font.Descent) * 0.5), bullet, 0));
01883                       }
01884                       else
01885                       {
01886                           isLoose = false;
01887                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
    Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent -
    context.Font.Descent * 0.5 - (context.Font.Ascent - context.Font.Descent) * 0.5), bullet, 0));
01888                       }
01889                   }
01890                   else
01891                   {
01892                       if (context.CurrentLine == null)
01893                       {
01894                           context.CurrentLine = new Line(context.Font.Ascent);
01895                       }
01896
01897                       Graphics bullet = new Graphics();
01898                       bullet.Scale(context.Font.FontSize, context.Font.FontSize);
01899                       this.Bullets[(context.ListDepth - 1) % this.Bullets.Count](bullet,
    context.Colour);
01900
01901                       if (list.IsLoose)
01902                       {
01903                           isLoose = true;
01904                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
    Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent +
    SpaceBeforeParagaph * context.Font.FontSize - context.Font.Descent * 0.5 - (context.Font.Ascent -
    context.Font.Descent) * 0.5), bullet, 0));
01905                       }
01906                       else
01907                       {
01908                           isLoose = false;
01909                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
    Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent -
    context.Font.Descent * 0.5 - (context.Font.Ascent - context.Font.Descent) * 0.5), bullet, 0));
01910                       }
01911                   }
01912               }
01913           }
01914
01915           foreach (Block block in listItem)
01916           {
01917               RenderBlock(block, ref context, ref graphics, newPageAction, isLoose || listItem ==
    listItem.Parent[0], isLoose || listItem == listItem.Parent.LastChild);
01918           }
01919
01920           prevContext.Cursor = context.Cursor;
01921           prevContext.BottomRight = context.BottomRight;
01922           prevContext.CurrentPage = context.CurrentPage;
01923           prevContext.CurrentLine = context.CurrentLine;
01924
01925           context = prevContext;
01926       }
01927
01928       private void RenderHTMLBlock(string html, bool isInline, ref MarkdownContext context, ref
    Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
01929       {
01930           if (!isInline)
01931           {
01932               double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
    context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeParagaph *
```

```
    context.Font.FontSize - context.Font.Descent);
01933
01934                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
    SpaceBeforeParagaph * context.Font.FontSize);
01935
01936                 if (context.CurrentLine == null)
01937                 {
01938                     context.CurrentLine = new Line(context.Font.Ascent);
01939                 }
01940             }
01941
01942             foreach (HtmlTag tag in HtmlTag.Parse(html))
01943             {
01944                 if (tag.Tag.Equals("img", StringComparison.OrdinalIgnoreCase) ||
    tag.Tag.Equals("image", StringComparison.OrdinalIgnoreCase))
01945                 {
01946                     RenderHTMLImage(tag, isInline, ref context, ref graphics, newPageAction);
01947                 }
01948                 else if (tag.Tag.Equals("br", StringComparison.OrdinalIgnoreCase))
01949                 {
01950                     RenderLineBreakInline(true, tag.Attributes.TryGetValue("type", out string
    typeValue) && typeValue.Equals("page", StringComparison.OrdinalIgnoreCase) && this.AllowPageBreak, ref
    context, ref graphics, newPageAction);
01951                 }
01952                 else if (tag.Tag.Equals("a"))
01953                 {
01954                     if (tag.Attributes.TryGetValue("name", out string anchorName))
01955                     {
01956                         if (!context.InternalAnchors.TryGetValue("#" + anchorName, out string anchor))
01957                         {
01958                             anchor = Guid.NewGuid().ToString("N");
01959                             context.InternalAnchors["#" + anchorName] = anchor;
01960                         }
01961
01962                         if (context.CurrentLine == null)
01963                         {
01964                             context.CurrentLine = new Line(0);
01965                         }
01966
01967                         double anchorHeight = this.BaseFontSize * Math.Max(1,
    this.HeaderFontSizeMultipliers.Max());
01968
01969                         context.CurrentLine.Fragments.Add(new RectangleFragment(new
    Point(context.Cursor.X, context.Cursor.Y - anchorHeight), new Size(10, anchorHeight),
    Colour.FromRgba(0, 0, 0, 0), anchor));
01970                     }
01971                 }
01972                 else
01973                 {
01974
01975                 }
01976             }
01977
01978             if (!isInline)
01979             {
01980                 if (context.CurrentLine != null)
01981                 {
01982                     context.CurrentLine.Render(ref graphics, ref context, newPageAction,
    this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
01983                     context.CurrentLine = null;
01984
01985                     context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
    SpaceAfterLine * context.Font.FontSize);
01986
01987                     context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
    context.Font.FontSize);
01988                 }
01989             }
01990         }
01991
01992         private void RenderHTMLImage(HtmlTag imgTag, bool isInline, ref MarkdownContext context, ref
    Graphics graphics, NewPageAction newPageAction)
01993         {
01994             if (imgTag != null)
01995             {
01996                 if (imgTag.Attributes.TryGetValue("src", out string imageSrc))
01997                 {
01998                     (string imageFile, bool wasDownloaded) = this.ImageUriResolver(imageSrc,
    this.BaseImageUri);
01999
02000                     Page imagePage = null;
02001
02002                     if (imageFile != null)
02003                     {
02004                         if (System.IO.Path.GetExtension(imageFile) == ".svg")
02005                         {
02006                             try
```

```
02007                           {
02008                               imagePage = VectSharp.SVG.Parser.FromFile(imageFile);
02009                           }
02010                           catch
02011                           {
02012                               imagePage = null;
02013                           }
02014                       }
02015                       else if (RasterImageLoader != null)
02016                       {
02017                           try
02018                           {
02019                               RasterImage raster = RasterImageLoader(imageFile);
02020                               imagePage = new Page(raster.Width, raster.Height);
02021                               imagePage.Graphics.DrawRasterImage(0, 0, raster, tag:  context.Tag);
02022                           }
02023                           catch
02024                           {
02025                               imagePage = null;
02026                           }
02027                       }
02028
02029
02030                       if (wasDownloaded)
02031                       {
02032                           System.IO.File.Delete(imageFile);
02033                           System.IO.Directory.Delete(System.IO.Path.GetDirectoryName(imageFile));
02034                       }
02035                   }
02036                   else if (imageSrc.StartsWith("data:"))
02037                   {
02038                       try
02039                       {
02040                           imagePage = VectSharp.SVG.Parser.ParseImageURI(imageSrc, false);
02041                       }
02042                       catch
02043                       {
02044
02045                           imagePage = null;
02046                       }
02047                   }
02048
02049                   if (imagePage != null)
02050                   {
02051                       double scaleX = 1;
02052                       double scaleY = 1;
02053
02054                       bool hasWidth = false;
02055                       bool hasHeight = false;
02056
02057                       if (imgTag.Attributes.TryGetValue("width", out string widthString) &&
      double.TryParse(widthString, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out double width))
02058                       {
02059                           hasWidth = true;
02060                           scaleX = width * this.ImageUnitMultiplier / imagePage.Width;
02061                       }
02062
02063                       if (imgTag.Attributes.TryGetValue("height", out string heightString) &&
      double.TryParse(heightString, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out double height))
02064                       {
02065                           hasHeight = true;
02066                           scaleY = height * this.ImageUnitMultiplier / imagePage.Height;
02067                       }
02068
02069                       if (hasWidth && !hasHeight)
02070                       {
02071                           scaleY = scaleX;
02072                       }
02073                       else if (hasHeight && !hasWidth)
02074                       {
02075                           scaleX = scaleY;
02076                       }
02077
02078                       scaleX *= this.ImageMultiplier;
02079                       scaleY *= this.ImageMultiplier;
02080
02081                       if (!isInline)
02082                       {
02083                           string alignValue;
02084
02085                           if (!imgTag.Attributes.TryGetValue("align", out alignValue))
02086                           {
02087                               alignValue = null;
02088                           }
02089
```

```
02090                           if (alignValue == "center")
02091                           {
02092                               if (context.CurrentLine != null)
02093                               {
02094                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y);
02095                                   context.CurrentLine = null;
02096                               }
02097
02098                               double minX = context.GetMinX(context.Cursor.Y +
      context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height);
02099                               double maxX = context.GetMaxX(context.Cursor.Y +
      context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X);
02100
02101                               if (scaleX * imagePage.Width > maxX - minX)
02102                               {
02103                                   scaleX = (maxX - minX) / imagePage.Width;
02104                                   scaleY = scaleX;
02105                               }
02106
02107                               double finalY = context.Cursor.Y + scaleY * imagePage.Height;
02108
02109                               if (finalY + context.Translation.Y > this.PageSize.Height -
      this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
02110                               {
02111                                   newPageAction(ref context, ref graphics);
02112                                   finalY = context.Cursor.Y + scaleY * imagePage.Height;
02113                               }
02114
02115                               graphics.Save();
02116                               graphics.Translate((minX + maxX - scaleX * imagePage.Width) * 0.5,
      context.Cursor.Y);
02117                               graphics.Scale(scaleX, scaleY);
02118                               graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
02119                               graphics.DrawGraphics(0, 0, imagePage.Graphics);
02120
02121                               graphics.Restore();
02122
02123                               context.Cursor = new Point(0, finalY + SpaceAfterParagraph *
      context.Font.FontSize + SpaceAfterLine * context.Font.FontSize);
02124                           }
02125                           else if (alignValue == "right")
02126                           {
02127                               if (context.CurrentLine != null)
02128                               {
02129                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y);
02130                                   context.CurrentLine = null;
02131                               }
02132
02133                               double finalY = context.Cursor.Y + scaleY * imagePage.Height;
02134
02135                               if (finalY + context.Translation.Y > this.PageSize.Height -
      this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
02136                               {
02137                                   newPageAction(ref context, ref graphics);
02138                                   finalY = context.Cursor.Y + scaleY * imagePage.Height;
02139                               }
02140
02141                               graphics.Save();
02142                               graphics.Translate(this.PageSize.Width - this.Margins.Right -
      context.Translation.X - scaleX * imagePage.Width - context.MarginBottomRight.X, context.Cursor.Y);
02143                               graphics.Scale(scaleX, scaleY);
02144                               graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
02145
02146                               graphics.DrawGraphics(0, 0, imagePage.Graphics);
02147
02148                               graphics.Restore();
02149
02150                               context.ForbiddenAreasRight.Add((this.PageSize.Width -
      this.Margins.Right - scaleX * imagePage.Width - this.ImageSideMargin - context.MarginBottomRight.X,
      context.Cursor.Y + context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY *
      imagePage.Height));
02151                           }
02152                           else if (alignValue == "left")
02153                           {
02154                               if (context.CurrentLine != null)
02155                               {
02156                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y);
02157                                   context.CurrentLine = null;
02158                               }
02159
```

```
02160                                 double finalY = context.Cursor.Y + scaleY * imagePage.Height;
02161
02162                                 if (finalY + context.Translation.Y > this.PageSize.Height -
      this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
02163                                 {
02164                                     newPageAction(ref context, ref graphics);
02165                                     finalY = context.Cursor.Y + scaleY * imagePage.Height;
02166                                 }
02167
02168                                 graphics.Save();
02169                                 graphics.Translate(0, context.Cursor.Y);
02170                                 graphics.Scale(scaleX, scaleY);
02171                                 graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
02172
02173                                 graphics.DrawGraphics(0, 0, imagePage.Graphics);
02174
02175                                 graphics.Restore();
02176
02177                                 context.ForbiddenAreasLeft.Add((scaleX * imagePage.Width +
      context.Translation.X + this.ImageSideMargin, context.Cursor.Y + context.Translation.Y,
      context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height));
02178                             }
02179                             else
02180                             {
02181                                 isInline = true;
02182                             }
02183                         }
02184
02185                         if (isInline)
02186                         {
02187                             Graphics scaledImage = new Graphics();
02188                             scaledImage.Scale(scaleX, scaleY);
02189                             scaledImage.DrawGraphics(0, 0, imagePage.Graphics);
02190
02191                             if (context.CurrentLine == null)
02192                             {
02193                                 context.CurrentLine = new Line(context.Font.Ascent);
02194
02195                                 context.Cursor = new Point(0, context.Cursor.Y);
02196                                 double minX = context.GetMinX(context.Cursor.Y - scaleY *
      imagePage.Height, context.Cursor.Y);
02197                                 context.Cursor = new Point(minX, context.Cursor.Y);
02198                             }
02199
02200                             double currLineMaxX = context.GetMaxX(context.Cursor.Y - scaleY *
      imagePage.Height, context.Cursor.Y, this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X);
02201
02202                             double finalX = context.Cursor.X + imagePage.Width;
02203
02204                             if (finalX > currLineMaxX)
02205                             {
02206                                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
02207
02208                                 context.CurrentLine = new Line(context.Font.Ascent);
02209                                 context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent
      + SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
02210                                 currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
02211
02212                                 double minX = context.GetMinX(context.Cursor.Y - scaleY *
      imagePage.Height * scaleY, context.Cursor.Y);
02213
02214                                 context.Cursor = new Point(minX, context.Cursor.Y);
02215                             }
02216
02217                             context.CurrentLine.Fragments.Add(new GraphicsFragment(new
      Point(context.Cursor.X, context.Cursor.Y - scaleY * imagePage.Height), scaledImage, scaleY *
      imagePage.Height));
02218                             context.Cursor = new Point(context.Cursor.X + scaleX * imagePage.Width,
      context.Cursor.Y);
02219                         }
02220                     }
02221                 }
02222             }
02223         }
02224
02225         private void RenderTable(Table table, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction)
02226         {
02227             if (table.Count > 0)
02228             {
02229                 if (!table.IsValid())
02230                 {
02231                     table.NormalizeUsingMaxWidth();
```

```
02232                     table.NormalizeUsingHeaderRow();
02233                 }
02234
02235                 if (table.IsValid() && table.ColumnDefinitions?.Count > 0)
02236                 {
02237                     bool isGridTable = false;
02238
02239                     foreach (TableColumnDefinition def in table.ColumnDefinitions)
02240                     {
02241                         if (def.Width > 0)
02242                         {
02243                             isGridTable = true;
02244                             break;
02245                         }
02246                     }
02247
02248                     if (isGridTable)
02249                     {
02250                         int maxColumns = 0;
02251
02252                         foreach (TableRow row in table)
02253                         {
02254                             maxColumns = Math.Max(maxColumns, ((TableCell)row.Last()).ColumnIndex +
     1);
02255                         }
02256
02257                         if (table.ColumnDefinitions.Count > maxColumns)
02258                         {
02259                             table.ColumnDefinitions.RemoveRange(maxColumns,
     table.ColumnDefinitions.Count - maxColumns);
02260                         }
02261                     }
02262                     else
02263                     {
02264                         int maxColumns = 0;
02265
02266                         foreach (TableRow row in table)
02267                         {
02268                             maxColumns = Math.Max(maxColumns, row.Count);
02269                         }
02270
02271                         if (table.ColumnDefinitions.Count > maxColumns)
02272                         {
02273                             table.ColumnDefinitions.RemoveRange(maxColumns,
     table.ColumnDefinitions.Count - maxColumns);
02274                         }
02275                     }
02276
02277                     double[] columnWidths = new double[table.ColumnDefinitions.Count];
02278
02279                     if (table.ColumnDefinitions.Count == 0)
02280                     {
02281                         int columnCount = 0;
02282                         foreach (TableRow row in table)
02283                         {
02284                             columnCount = Math.Max(row.Count, columnCount);
02285                         }
02286
02287                         columnWidths = new double[columnCount];
02288                     }
02289
02290                     int missingColumns = columnWidths.Length;
02291
02292                     for (int i = 0; i < columnWidths.Length; i++)
02293                     {
02294                         columnWidths[i] = double.NaN;
02295                     }
02296
02297                     double remainingPerc = 1;
02298
02299                     for (int i = 0; i < table.ColumnDefinitions.Count; i++)
02300                     {
02301                         if (table.ColumnDefinitions[i].Width > 0)
02302                         {
02303                             missingColumns--;
02304                             remainingPerc -= table.ColumnDefinitions[i].Width / 100.0;
02305                             columnWidths[i] = table.ColumnDefinitions[i].Width / 100.0;
02306                         }
02307                     }
02308
02309                     if (missingColumns > 0)
02310                     {
02311                         remainingPerc /= missingColumns;
02312                         for (int i = 0; i < columnWidths.Length; i++)
02313                         {
02314                             if (double.IsNaN(columnWidths[i]))
02315                             {
```

```
02316                                 columnWidths[i] = remainingPerc;
02317                             }
02318                         }
02319                     }
02320 
02321                     double maxX = context.GetMaxX(context.Cursor.Y, context.Cursor.Y,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X);
02322 
02323                     for (int i = 0; i < columnWidths.Length; i++)
02324                     {
02325                         columnWidths[i] *= maxX;
02326                     }
02327 
02328                     int index = 0;
02329 
02330                     foreach (TableRow row in table)
02331                     {
02332                         RenderTableRow(row, columnWidths, index == table.Count - 1, ref context, ref
      graphics, newPageAction);
02333                         index++;
02334                     }
02335 
02336                     context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      SpaceAfterParagraph * context.Font.FontSize);
02337                 }
02338             }
02339         }
02340 
02341         private void RenderTableRow(TableRow row, double[] columnWidths, bool isLastRow, ref
      MarkdownContext context, ref Graphics graphics, NewPageAction newPageAction)
02342         {
02343             if (context.CurrentLine == null)
02344             {
02345                 context.CurrentLine = new Line(0);
02346             }
02347 
02348             int index = 0;
02349 
02350             foreach (TableCell cell in row)
02351             {
02352                 if (cell.ColumnIndex < 0)
02353                 {
02354                     cell.ColumnIndex = index;
02355                     index += cell.ColumnSpan;
02356                 }
02357                 else
02358                 {
02359                     index = cell.ColumnIndex + cell.ColumnSpan;
02360                 }
02361             }
02362 
02363             double maxHeight = 0;
02364             double startX = context.Cursor.X;
02365 
02366             MarkdownContext prevContext = context.Clone();
02367 
02368             if (row.IsHeader)
02369             {
02370                 context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
02371             }
02372 
02373 
02374             foreach (TableCell cell in row)
02375             {
02376                 double cellWidth = 0;
02377 
02378                 for (int i = 0; i < cell.ColumnSpan && cell.ColumnIndex + i < columnWidths.Length;
      i++)
02379                 {
02380                     cellWidth += columnWidths[cell.ColumnIndex + i];
02381                 }
02382 
02383                 Page cellPage = RenderTableCell(cell, cellWidth, ref context, ref graphics,
      newPageAction);
02384 
02385                 double prevMaxHeight = maxHeight;
02386                 maxHeight = Math.Max(maxHeight, cellPage.Height);
02387 
02388                 context.CurrentLine.Fragments.Add(new GraphicsFragment(new Point(context.Cursor.X,
      context.Cursor.Y - cellPage.Height), cellPage.Graphics, cellPage.Height));
02389 
02390                 context.Cursor = new Point(context.Cursor.X + cellWidth, context.Cursor.Y);
02391             }
02392 
02393             if (this.TableVAlign == VerticalAlignment.Top)
02394             {
02395                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
```

```
02396                 {
02397                     context.CurrentLine.Fragments[i].Translate(0, -maxHeight +
      ((GraphicsFragment)context.CurrentLine.Fragments[i]).Ascent);
02398                 }
02399             }
02400             else if (this.TableVAlign == VerticalAlignment.Middle)
02401             {
02402                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
02403                 {
02404                     context.CurrentLine.Fragments[i].Translate(0, (-maxHeight +
      ((GraphicsFragment)context.CurrentLine.Fragments[i]).Ascent) * 0.5);
02405                 }
02406             }
02407 
02408             context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(startX,
      context.Cursor.Y), new Point(columnWidths.Sum() + startX, context.Cursor.Y), row.IsHeader ?
      this.TableHeaderRowSeparatorColour :  this.TableRowSeparatorColour, row.IsHeader ?
      this.TableHeaderRowSeparatorThickness :  this.TableHeaderSeparatorThickness, context.Tag));
02409 
02410             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y);
02411             context.CurrentLine = null;
02412 
02413             context.Cursor = new Point(startX, context.Cursor.Y);
02414 
02415             context.Cursor = new Point(startX, context.Cursor.Y + SpaceAfterLine *
      context.Font.FontSize + (row.IsHeader ?  this.TableHeaderRowSeparatorThickness :
      this.TableHeaderSeparatorThickness));
02416 
02417             prevContext.Cursor = context.Cursor;
02418             prevContext.BottomRight = context.BottomRight;
02419             prevContext.CurrentPage = context.CurrentPage;
02420             prevContext.CurrentLine = context.CurrentLine;
02421 
02422             context = prevContext;
02423         }
02424 
02425         private Page RenderTableCell(TableCell cell, double cellWidth, ref MarkdownContext context,
      ref Graphics graphics, NewPageAction newPageAction)
02426         {
02427             MarkdownRenderer clonedRenderer = this.Clone();
02428             clonedRenderer.PageSize = new Size(cellWidth, double.PositiveInfinity);
02429             clonedRenderer.Margins = this.TableCellMargins;
02430 
02431             MarkdownContext clonedContext = context.Clone();
02432             clonedContext.Translation = new Point(0, 0);
02433             clonedContext.Cursor = new Point(0, 0);
02434             clonedContext.BottomRight = new Point(0, 0);
02435             clonedContext.CurrentLine = null;
02436             clonedContext.CurrentPage = null;
02437             clonedContext.ForbiddenAreasLeft = new List<(double MinX, double MinY, double MaxY)>();
02438             clonedContext.ForbiddenAreasRight = new List<(double MinX, double MinY, double MaxY)>();
02439 
02440             Page cellPage = clonedRenderer.RenderSubDocument(cell, ref clonedContext).Pages[0];
02441 
02442             cellPage.Crop(new Point(0, 0), new Size(cellWidth, clonedContext.BottomRight.Y +
      this.TableCellMargins.Bottom));
02443 
02444             return cellPage;
02445         }
02446 
02447         static (int, string)[] RomanNumbers = new (int, string)[] { (1000, "M"), (900, "CM"), (500,
      "D"), (400, "CD"), (100, "C"), (90, "XC"), (50, "L"), (40, "XL"), (10, "X"), (9, "IX"), (5, "V"), (4,
      "IV"), (1, "I") };
02448 
02449         private static string GetRomanNumeral(int number)
02450         {
02451             StringBuilder tbr = new StringBuilder();
02452 
02453             for (int i = 0; i < RomanNumbers.Length; i++)
02454             {
02455                 while (number >= RomanNumbers[i].Item1)
02456                 {
02457                     tbr.Append(RomanNumbers[i].Item2);
02458                     number -= RomanNumbers[i].Item1;
02459                 }
02460             }
02461 
02462             return tbr.ToString();
02463         }
02464     }
02465 }
```

8.12 SyntaxHighlighting.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Highlight;
00019 using Highlight.Engines;
00020 using Highlight.Patterns;
00021 using System;
00022 using System.Collections.Generic;
00023 using System.Linq;
00024 using System.Text;
00025 using System.Text.RegularExpressions;
00026
00027 namespace VectSharp.Markdown
00028 {
00029 /// <summary>
00030 /// Represents a string with associated formatting information.
00031 /// </summary>
00032     public struct FormattedString
00033     {
00034 /// <summary>
00035 /// The text represented by this object.
00036 /// </summary>
00037         public string Text { get; }
00038
00039 /// <summary>
00040 /// The colour of the text.
00041 /// </summary>
00042         public Colour Colour { get; }
00043
00044 /// <summary>
00045 /// Whether the text should be rendered as bold or not.
00046 /// </summary>
00047         public bool IsBold { get; }
00048
00049 /// <summary>
00050 /// Whether the text should be rendered as italic or not.
00051 /// </summary>
00052         public bool IsItalic { get; }
00053
00054 /// <summary>
00055 /// Creates a new <see cref="FormattedString"/> instance.
00056 /// </summary>
00057 /// <param name="text">The text of the object.</param>
00058 /// <param name="colour">The colour of the text.</param>
00059 /// <param name="isBold">Whether the text should be rendered as bold or not.</param>
00060 /// <param name="isItalic">Whether the text should be rendered as italic or not.</param>
00061         public FormattedString(string text, Colour colour, bool isBold, bool isItalic)
00062         {
00063             this.Text = text;
00064             this.Colour = colour;
00065             this.IsBold = isBold;
00066             this.IsItalic = isItalic;
00067         }
00068     }
00069
00070 /// <summary>
00071 /// Contains methods to perform syntax highlighting.
00072 /// </summary>
00073     public static class SyntaxHighlighter
00074     {
00075         private static Dictionary<string, string[]> LanguageAliases = new Dictionary<string,
      string[]>()
00076         {
00077             { "ASPX", new string[] { "ASP.NET", "aspx", "aspx-vb" } },
00078             { "C", new string[] { } },
00079             { "C++", new string[] { "cpp" } },
00080             { "C#", new string[] { "csharp" } },
00081             { "COBOL", new string[] { } },
00082             { "Eiffel", new string[] { } },
00083             { "Fortran", new string[] { } },
00084             { "Haskell", new string[] { } },
```

```
00085             { "HTML", new string[] { "xhtml" } },
00086             { "Java", new string[] { } },
00087             { "JavaScript", new string[] { "js", "node" } },
00088             { "Mercury", new string[] { } },
00089             { "MSIL", new string[] { } },
00090             { "Pascal", new string[] { "delphi", "objectpascal" } },
00091             { "Perl", new string[] { "cperl" } },
00092             { "PHP", new string[] { "inc" } },
00093             { "Python", new string[] { "python3", "rusthon" } },
00094             { "Ruby", new string[] { "jruby", "macruby", "rake", "rb", "rbx" } },
00095             { "SQL", new string[] { } },
00096             { "Visual Basic", new string[] { "vba", "vb6", "visual basic 6", "visual basic for
      applications" } },
00097             { "VBScript", new string[] { } },
00098             { "VB.NET", new string[] { "Visual Basic .NET", "vbnet", "vb .net" } },
00099             { "XML", new string[] { "rss", "xsd", "wsdl" } },
00100         };
00101
00102         private static string GetLanguage(string language)
00103         {
00104             foreach (KeyValuePair<string, string[]> element in LanguageAliases)
00105             {
00106                 if (element.Key.Equals(language, StringComparison.OrdinalIgnoreCase))
00107                 {
00108                     return element.Key;
00109                 }
00110
00111                 foreach (string alias in element.Value)
00112                 {
00113                     if (alias.Equals(language, StringComparison.OrdinalIgnoreCase))
00114                     {
00115                         return element.Key;
00116                     }
00117                 }
00118             }
00119
00120             return null;
00121         }
00122
00123 /// <summary>
00124 /// Performs syntax highlighting for a specified language on some source code.
00125 /// </summary>
00126 /// <param name="sourceCode">The source code to be highlighted.</param>
00127 /// <param name="language">The name of the language to use for the highlighting.</param>
00128 /// <returns>A list of lists of <see cref="FormattedString"/>s.  Each list of <see
      cref="FormattedString"/>s represents a line.</returns>
00129         public static List<List<FormattedString» GetSyntaxHighlightedLines(string sourceCode, string
      language)
00130         {
00131             language = GetLanguage(language);
00132
00133             if (string.IsNullOrEmpty(language))
00134             {
00135                 return null;
00136             }
00137
00138             HighlighterEngine engine = new HighlighterEngine();
00139
00140             Highlighter highlighter = new Highlighter(engine);
00141
00142             highlighter.Highlight(language, sourceCode);
00143
00144             List<List<FormattedString» tbr = new List<List<FormattedString»();
00145
00146             List<FormattedString> currentLine = new List<FormattedString>();
00147
00148             for (int i = 0; i < engine.HighlightedSpans.Count; i++)
00149             {
00150                 string[] split = engine.HighlightedSpans[i].Text.Replace("\r", "").Split('\n');
00151
00152                 for (int j = 0; j < split.Length; j++)
00153                 {
00154                     currentLine.Add(new FormattedString(split[j], engine.HighlightedSpans[i].Colour,
      engine.HighlightedSpans[i].IsBold, engine.HighlightedSpans[i].IsItalic));
00155
00156                     if (j < split.Length - 1)
00157                     {
00158                         tbr.Add(currentLine);
00159                         currentLine = new List<FormattedString>();
00160                     }
00161                 }
00162             }
00163
00164             tbr.Add(currentLine);
00165
00166             return tbr;
00167         }
```

```
00168     }
00169
00170     internal class HighlighterEngine :  Engine, IEngine
00171     {
00172         private const RegexOptions DefaultRegexOptions = RegexOptions.ExplicitCapture |
      RegexOptions.IgnorePatternWhitespace;
00173
00174         public List<FormattedString> HighlightedSpans { get; } = new List<FormattedString>();
00175
00176         protected override string ProcessBlockPatternMatch(Definition definition, BlockPattern
      pattern, Match match)
00177         {
00178             if (!string.IsNullOrEmpty(pattern.RawRegex))
00179             {
00180                 Match reMatch = Regex.Match(match.Value, pattern.RawRegex);
00181
00182                 if (reMatch.Groups.Count > 1)
00183                 {
00184                     HighlightedSpans.Add(new FormattedString(match.Value.Substring(0,
      reMatch.Groups[1].Index), Colours.Black, false, false));
00185                     HighlightedSpans.Add(new FormattedString(reMatch.Groups[1].Value,
      pattern.Style.Colors.ForeColor, pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00186                     HighlightedSpans.Add(new
      FormattedString(match.Value.Substring(reMatch.Groups[1].Index + reMatch.Groups[1].Length),
      Colours.Black, false, false));
00187                 }
00188                 else
00189                 {
00190                     HighlightedSpans.Add(new FormattedString(match.Value,
      pattern.Style.Colors.ForeColor, pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00191                 }
00192             }
00193             else
00194             {
00195                 HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00196             }
00197             return match.Value;
00198         }
00199
00200         protected override string ProcessMarkupPatternMatch(Definition definition, MarkupPattern
      pattern, Match match)
00201         {
00202             HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00203             return match.Value;
00204         }
00205
00206         protected override string ProcessWordPatternMatch(Definition definition, WordPattern pattern,
      Match match)
00207         {
00208             HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00209             return match.Value;
00210         }
00211
00212         string IEngine.Highlight(Definition definition, string input)
00213         {
00214             if (definition == null)
00215             {
00216                 throw new ArgumentNullException("definition");
00217             }
00218
00219             var output = PreHighlight(definition, input);
00220             output = HighlightUsingRegex(definition, output);
00221             output = PostHighlight(definition, output);
00222
00223             return output;
00224         }
00225
00226         private string HighlightUsingRegex(Definition definition, string input)
00227         {
00228             var regexOptions = GetRegexOptions(definition);
00229             var evaluator = GetMatchEvaluator(definition);
00230             var regexPattern = definition.GetRegexPattern();
00231
00232             int currentIndex = 0;
00233
00234             foreach (Match match in Regex.Matches(input, regexPattern))
00235             {
00236                 if (match.Index > currentIndex)
00237                 {
00238                     this.HighlightedSpans.Add(new FormattedString(input.Substring(currentIndex,
      match.Index - currentIndex), Colours.Black, false, false));
00239                 }
00240
00241                 currentIndex = match.Index + match.Length;
```

```
00242
00243                 evaluator(match);
00244             }
00245
00246             if (currentIndex < input.Length)
00247             {
00248                 this.HighlightedSpans.Add(new FormattedString(input.Substring(currentIndex,
      input.Length - currentIndex), Colours.Black, false, false));
00249             }
00250
00251             return null;
00252         }
00253
00254         private RegexOptions GetRegexOptions(Definition definition)
00255         {
00256             if (definition.CaseSensitive)
00257             {
00258                 return DefaultRegexOptions | RegexOptions.IgnoreCase;
00259             }
00260
00261             return DefaultRegexOptions;
00262         }
00263
00264         private string ElementMatchHandler(Definition definition, Match match)
00265         {
00266             if (definition == null)
00267             {
00268                 throw new ArgumentNullException("definition");
00269             }
00270             if (match == null)
00271             {
00272                 throw new ArgumentNullException("match");
00273             }
00274
00275             var pattern = definition.Patterns.First(x => match.Groups[x.Key].Success).Value;
00276             if (pattern != null)
00277             {
00278                 if (pattern is BlockPattern)
00279                 {
00280                     return ProcessBlockPatternMatch(definition, (BlockPattern)pattern, match);
00281                 }
00282                 if (pattern is MarkupPattern)
00283                 {
00284                     return ProcessMarkupPatternMatch(definition, (MarkupPattern)pattern, match);
00285                 }
00286                 if (pattern is WordPattern)
00287                 {
00288                     return ProcessWordPatternMatch(definition, (WordPattern)pattern, match);
00289                 }
00290             }
00291
00292             return match.Value;
00293         }
00294
00295         private MatchEvaluator GetMatchEvaluator(Definition definition)
00296         {
00297             return match => ElementMatchHandler(definition, match);
00298         }
00299     }
00300 }
```

8.13 Word.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
```

```
00021
00022 namespace VectSharp.Markdown
00023 {
00024     internal class Word
00025     {
00026         public char PrecedingWhitespace { get; }
00027         public int WhitespaceCount { get; }
00028         public string Text { get; }
00029
00030         public Font.DetailedFontMetrics Metrics { get; }
00031
00032         private Word(char precedingWhitespace, int whitespaceCount, string text, Font font)
00033         {
00034             this.PrecedingWhitespace = precedingWhitespace;
00035             this.WhitespaceCount = whitespaceCount;
00036             this.Text = text;
00037
00038             if (!string.IsNullOrEmpty(text))
00039             {
00040                 this.Metrics = font.MeasureTextAdvanced(text);
00041             }
00042             else
00043             {
00044                 this.Metrics = font.MeasureTextAdvanced(" ");
00045             }
00046         }
00047
00048         private static Dictionary<FontFamily, double> SpaceWidths { get; } = new
      Dictionary<FontFamily, double>();
00049
00050         private static IEnumerable<Word> SplitWord(Word word, Font font, double maxWidth)
00051         {
00052             if (!SpaceWidths.TryGetValue(font.FontFamily, out double spaceWidth))
00053             {
00054                 spaceWidth = font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0;
00055                 SpaceWidths[font.FontFamily] = spaceWidth;
00056             }
00057
00058             spaceWidth *= font.FontSize;
00059
00060             while (!string.IsNullOrEmpty(word.Text) && word.Metrics.Width +
      word.Metrics.LeftSideBearing + word.Metrics.RightSideBearing + spaceWidth * word.WhitespaceCount *
      (word.PrecedingWhitespace == '\t' ?  4 :  1) > maxWidth)
00061             {
00062                 int minIndex = 1;
00063                 int maxIndex = word.Text.Length;
00064
00065                 while (maxIndex - minIndex > 1)
00066                 {
00067                     int index = (minIndex + maxIndex) / 2;
00068
00069                     double width = font.MeasureText(word.Text.Substring(0, index)).Width;
00070
00071                     if (width > maxWidth)
00072                     {
00073                         maxIndex = index;
00074                     }
00075                     else if (width < maxWidth)
00076                     {
00077                         minIndex = index;
00078                     }
00079                     else
00080                     {
00081                         minIndex = index;
00082                         maxIndex = index;
00083                     }
00084                 }
00085
00086                 Word newWord = new Word(word.PrecedingWhitespace, word.WhitespaceCount,
      word.Text.Substring(0, minIndex), font);
00087                 yield return newWord;
00088
00089                 word = new Word('\0', 0, word.Text.Substring(minIndex), font);
00090             }
00091
00092             yield return word;
00093         }
00094
00095         public static IEnumerable<Word> GetWords(string text, Font font, double maxWidth)
00096         {
00097             if (!SpaceWidths.TryGetValue(font.FontFamily, out double spaceWidth))
00098             {
00099                 spaceWidth = font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0;
00100                 SpaceWidths[font.FontFamily] = spaceWidth;
00101             }
00102
00103             spaceWidth *= font.FontSize;
```

```
00104
00105             foreach (Word w in GetWordsInternal(text, font))
00106             {
00107                 if (w.Metrics.Width + w.Metrics.LeftSideBearing + w.Metrics.RightSideBearing +
      spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ?  4 :  1) > maxWidth)
00108                 {
00109                     foreach (Word w2 in SplitWord(w, font, maxWidth))
00110                     {
00111                         yield return w2;
00112                     }
00113                 }
00114                 else
00115                 {
00116                     yield return w;
00117                 }
00118             }
00119         }
00120
00121         private static IEnumerable<Word> GetWordsInternal(string text, Font font)
00122         {
00123             StringBuilder currWord = new StringBuilder();
00124
00125             char currWhitespace = '\0';
00126             int whitespaceCount = 0;
00127
00128             for (int i = 0; i < text.Length; i++)
00129             {
00130                 if (char.IsWhiteSpace(text[i]))
00131                 {
00132                     if (currWord.Length > 0)
00133                     {
00134                         yield return new Word(currWhitespace, whitespaceCount, currWord.ToString(),
      font);
00135
00136                         currWord.Clear();
00137                         currWhitespace = text[i];
00138                         whitespaceCount = 1;
00139                     }
00140                     else if (currWhitespace == text[i])
00141                     {
00142                         whitespaceCount++;
00143                     }
00144                     else
00145                     {
00146                         yield return new Word(currWhitespace, whitespaceCount, null, font);
00147
00148                         currWhitespace = text[i];
00149                         whitespaceCount = 1;
00150                     }
00151                 }
00152                 else
00153                 {
00154                     currWord.Append(text[i]);
00155                 }
00156             }
00157
00158             if (currWord.Length > 0)
00159             {
00160                 yield return new Word(currWhitespace, whitespaceCount, currWord.ToString(), font);
00161
00162                 currWord.Clear();
00163             }
00164             else if (whitespaceCount > 0)
00165             {
00166                 yield return new Word(currWhitespace, whitespaceCount, null, font);
00167             }
00168         }
00169     }
00170 }
```

8.14 ImageCache.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia.Controls.ApplicationLifetimes;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022
00023 namespace VectSharp.MarkdownCanvas
00024 {
00025     internal static class ImageCache
00026     {
00027         private const string imageCacheId = "bb5a724e-93f7-431d-87c3-53f31d8da16e";
00028
00029         private static string imageCacheFolder;
00030
00031         private static Dictionary<string, string> imageCache;
00032
00033         static ImageCache()
00034         {
00035             imageCacheFolder = Path.Combine(Path.GetTempPath(), imageCacheId);
00036             Directory.CreateDirectory(imageCacheFolder);
00037             imageCache = new Dictionary<string, string>();
00038         }
00039
00040         private static bool exitHandlerSet = false;
00041         public static void SetExitEventHandler()
00042         {
00043             if (!exitHandlerSet)
00044             {
00045                 if (Avalonia.Application.Current.ApplicationLifetime is IControlledApplicationLifetime
      lifetime)
00046                 {
00047                     lifetime.Exit += (s, e) =>
00048                     {
00049                         try
00050                         {
00051                             Directory.Delete(imageCacheFolder, true);
00052                         }
00053                         catch
00054                         {
00055                             try
00056                             {
00057                                 foreach (string sr in Directory.GetFiles(imageCacheFolder, "*.*"))
00058                                 {
00059                                     try
00060                                     {
00061                                         File.Delete(sr);
00062                                     }
00063                                     catch { }
00064                                 }
00065                             }
00066                             catch { }
00067                         }
00068                     };
00069                 }
00070                 exitHandlerSet = true;
00071             }
00072         }
00073
00074         public static (string, bool) ImageUriResolver(string imageUri, string baseUriString)
00075         {
00076             if (!imageCache.TryGetValue(baseUriString + "|||" + imageUri, out string cachedImage))
00077             {
00078                 (string imagePath, bool wasDownloaded) =
      VectSharp.Markdown.HTTPUtils.ResolveImageURI(imageUri, baseUriString);
00079
00080                 if (!string.IsNullOrEmpty(imagePath) && File.Exists(imagePath))
00081                 {
00082                     string id = Guid.NewGuid().ToString();
00083
00084                     cachedImage = Path.Combine(imageCacheFolder, id + Path.GetExtension(imagePath));
00085
00086                     if (wasDownloaded)
00087                     {
00088                         if (!Directory.Exists(imageCacheFolder))
00089                         {
00090                             Directory.CreateDirectory(imageCacheFolder);
00091                         }
00092
00093                         File.Move(imagePath, cachedImage);
00094                         Directory.Delete(Path.GetDirectoryName(imagePath));
00095                     }
00096                     else
```

```
00097                     {
00098                         File.Copy(imagePath, cachedImage);
00099                     }
00100 
00101                     imageCache[baseUriString + "|||" + imageUri] = cachedImage;
00102                 }
00103                 else
00104                 {
00105                     cachedImage = null;
00106                 }
00107             }
00108 
00109             return (cachedImage, false);
00110         }
00111     }
00112 }
```

8.15 MarkdownCanvas.axaml.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Avalonia;
00019 using Avalonia.Controls;
00020 using Avalonia.Controls.ApplicationLifetimes;
00021 using Avalonia.Markup.Xaml;
00022 using Avalonia.Media;
00023 using Avalonia.Threading;
00024 using Markdig;
00025 using Markdig.Syntax;
00026 using System;
00027 using System.Collections.Generic;
00028 using VectSharp.Canvas;
00029 using VectSharp.Markdown;
00030 
00031 namespace VectSharp.MarkdownCanvas
00032 {
00033 /// <summary>
00034 /// A control to display a Markdown document in an Avalonia application.
00035 /// </summary>
00036     public partial class MarkdownCanvasControl :  UserControl
00037     {
00038 /// <summary>
00039 /// Defines the <see cref="MaxRenderWidth"/> property.
00040 /// </summary>
00041         public static readonly StyledProperty<double> MaxRenderWidthProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth),
      double.PositiveInfinity);
00042 
00043 /// <summary>
00044 /// The maximum width for the rendered document.  This will be used even if the control's client area
      is larger than this (the alignment of the document within the controll will depend on the control's
      <see cref="ContentControl.HorizontalContentAlignment"/>).
00045 /// </summary>
00046         public double MaxRenderWidth
00047         {
00048             get { return GetValue(MaxRenderWidthProperty); }
00049             set { SetValue(MaxRenderWidthProperty, value); }
00050         }
00051 
00052 /// <summary>
00053 /// Defines the <see cref="MinRenderWidth"/> property.
00054 /// </summary>
00055         public static readonly StyledProperty<double> MinRenderWidthProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth), 200);
00056 
00057 /// <summary>
00058 /// The minimum width for the rendered document.  If the control's client area is smaller than this,
      the horizontal scroll bar will be activated.
```

```
00059 /// </summary>
00060         public double MinRenderWidth
00061         {
00062             get { return GetValue(MinRenderWidthProperty); }
00063             set { SetValue(MinRenderWidthProperty, value); }
00064         }
00065 
00066 /// <summary>
00067 /// Defines the <see cref="MinVariation"/> property.
00068 /// </summary>
00069         public static readonly StyledProperty<double> MinVariationProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10);
00070 
00071 /// <summary>
00072 /// The minimum width variation that triggers a document reflow.  If the control is resized, but the
      width changes by less than this amount, the document is not re-drawn.
00073 /// </summary>
00074         public double MinVariation
00075         {
00076             get { return GetValue(MinVariationProperty); }
00077             set { SetValue(MinVariationProperty, value); }
00078         }
00079 
00080 /// <summary>
00081 /// Defines the <see cref="DocumentSource"/> property.
00082 /// </summary>
00083         public static readonly StyledProperty<string> DocumentSourceProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource));
00084 
00085 /// <summary>
00086 /// Sets the currently displayed document from Markdown source.
00087 /// </summary>
00088         public string DocumentSource
00089         {
00090             set { SetValue(DocumentSourceProperty, value); }
00091         }
00092 
00093 /// <summary>
00094 /// Defines the <see cref="Document"/> property.
00095 /// </summary>
00096         public static readonly StyledProperty<MarkdownDocument> DocumentProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document));
00097 
00098 /// <summary>
00099 /// Gets or sets the currently displayed <see cref="MarkdownDocument"/>.
00100 /// </summary>
00101         public MarkdownDocument Document
00102         {
00103             get { return GetValue(DocumentProperty); }
00104             set { SetValue(DocumentProperty, value); }
00105         }
00106 
00107 /// <summary>
00108 /// Defines the <see cref="TextConversionOption"/> property.
00109 /// </summary>
00110         public static readonly StyledProperty<AvaloniaContextInterpreter.TextOptions>
      TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl,
      AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOption),
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary);
00111 
00112 /// <summary>
00113 /// Gets or sets the value that determines whether text items should be converted into paths when
      drawing.
00114 /// Setting this to <see cref="AvaloniaContextInterpreter.TextOptions.NeverConvert"/> will improve
      performance if you are using custom fonts, but may cause unexpected results unless the font families
      being used are of type <see cref="ResourceFontFamily"/>.
00115 /// </summary>
00116         public AvaloniaContextInterpreter.TextOptions TextConversionOption
00117         {
00118             get { return GetValue(TextConversionOptionsProperty); }
00119             set { SetValue(TextConversionOptionsProperty, value); }
00120         }
00121 
00122 /// <summary>
00123 /// The <see cref="MarkdownRenderer"/> used to render the <see cref="Document"/>.  You can use the
      properties of this object to customise the rendering.  Note that setting the <see
      cref="Avalonia.Controls.Primitives.TemplatedControl.FontSize"/> of the <see
      cref="MarkdownCanvasControl"/> will propagate to the <see cref="Renderer"/>'s <see
      cref="MarkdownRenderer.BaseFontSize"/>.
00124 /// </summary>
00125         public MarkdownRenderer Renderer { get; }
00126 
00127         private double lastRenderedWidth = double.NaN;
00128 
00129         private bool initialized = false;
00130 
00131 /// <summary>
```

```
00132 /// Initialises a new <see cref="MarkdownCanvasControl"/>.
00133 /// </summary>
00134         public MarkdownCanvasControl()
00135         {
00136             InitializeComponent();
00137             this.Renderer = new MarkdownRenderer() { BaseFontSize = this.FontSize, Margins = new
      Margins(10, 10, 10, 10), ImageUriResolver = ImageCache.ImageUriResolver };
00138             this.initialized = true;
00139             ImageCache.SetExitEventHandler();
00140         }
00141
00142 /// <inheritdoc/>
00143         protected override void OnPropertyChanged(AvaloniaPropertyChangedEventArgs change)
00144         {
00145             if (change.Property != ContentProperty || !initialized)
00146             {
00147                 base.OnPropertyChanged(change);
00148
00149                 if (this.initialized)
00150                 {
00151                     if (change.Property == MarkdownCanvasControl.BoundsProperty || change.Property ==
      MarkdownCanvasControl.MinRenderWidthProperty || change.Property ==
      MarkdownCanvasControl.MaxRenderWidthProperty || change.Property ==
      MarkdownCanvasControl.MinVariationProperty)
00152                     {
00153                         Render();
00154                     }
00155                     else if (change.Property == MarkdownCanvasControl.DocumentProperty)
00156                     {
00157                         forcedRerender = true;
00158                         Render();
00159                     }
00160                     else if (change.Property == MarkdownCanvasControl.FontSizeProperty)
00161                     {
00162                         this.Renderer.BaseFontSize = this.FontSize;
00163                         forcedRerender = true;
00164                         Render();
00165                     }
00166                     else if (change.Property == MarkdownCanvasControl.DocumentSourceProperty)
00167                     {
00168                         MarkdownDocument document =
      Markdig.Markdown.Parse(change.GetNewValue<string>(), new
      MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGenericAttributes().UseAutoIdentifiers
00169                         this.Document = document;
00170                     }
00171                     else if (change.Property ==
      MarkdownCanvasControl.VerticalContentAlignmentProperty)
00172                     {
00173                         this.FindControl<ScrollViewer>("ScrollViewer").VerticalContentAlignment =
      this.VerticalContentAlignment;
00174                     }
00175                     else if (change.Property ==
      MarkdownCanvasControl.HorizontalContentAlignmentProperty)
00176                     {
00177                         this.FindControl<ScrollViewer>("ScrollViewer").HorizontalContentAlignment =
      this.HorizontalContentAlignment;
00178                     }
00179                 }
00180             }
00181             else
00182             {
00183                 initialized = false;
00184                 this.Content = change.OldValue;
00185                 initialized = true;
00186             }
00187         }
00188
00189         private bool forcedRerender = false;
00190
00191         private void Render()
00192         {
00193             if (Document != null)
00194             {
00195                 double width = Math.Min(MaxRenderWidth, Math.Max(MinRenderWidth, this.Bounds.Width -
      MinVariation - 13));
00196
00197                 if (forcedRerender || double.IsNaN(lastRenderedWidth) || width != lastRenderedWidth &&
      width < lastRenderedWidth - MinVariation || width > lastRenderedWidth + MinVariation)
00198                 {
00199                     Page pag;
00200                     Dictionary<string, string> linkDestinations;
00201
00202                     try
00203                     {
00204                         pag = Renderer.RenderSinglePage(this.Document, width, out linkDestinations);
00205                     }
00206                     catch
```

```
00207                     {
00208                         pag = new Page(width, 0);
00209                         linkDestinations = new Dictionary<string, string>();
00210                     }
00211 
00212                     Dictionary<string, Delegate> taggedActions = new Dictionary<string, Delegate>();
00213                     Dictionary<string, Avalonia.Point> linkDestinationPoints = new Dictionary<string,
      Avalonia.Point>();
00214 
00215                     foreach (KeyValuePair<string, string> linkDestination in linkDestinations)
00216                     {
00217                         string url = linkDestination.Value;
00218 
00219                         taggedActions.Add(linkDestination.Key, (Func<RenderAction,
      IEnumerable<RenderAction>>)(act =>
00220                         {
00221                             act.PointerEntered += (s, e) =>
00222                             {
00223                                 act.Parent.Cursor = new
      Avalonia.Input.Cursor(Avalonia.Input.StandardCursorType.Hand);
00224                             };
00225 
00226                             act.PointerExited += (s, e) =>
00227                             {
00228                                 act.Parent.Cursor = new
      Avalonia.Input.Cursor(Avalonia.Input.StandardCursorType.Arrow);
00229                             };
00230 
00231                             act.PointerPressed += (s, e) =>
00232                             {
00233                                 if (url.StartsWith("#"))
00234                                 {
00235                                     if (linkDestinationPoints.TryGetValue(url.Substring(1), out
      Avalonia.Point target))
00236                                     {
00237                                         ScrollViewer scrollViewer =
      this.FindControl<ScrollViewer>("ScrollViewer");
00238 
00239                                         scrollViewer.Offset = new
      Vector(Math.Max(Math.Min(scrollViewer.Offset.X, target.X), target.X - scrollViewer.Viewport.Width),
      target.Y);
00240                                     }
00241                                 }
00242                                 else
00243                                 {
00244                                     System.Diagnostics.Process.Start(new
      System.Diagnostics.ProcessStartInfo() { FileName = url, UseShellExecute = true });
00245                                 }
00246                             };
00247 
00248                             if (act.ActionType == RenderAction.ActionTypes.Path)
00249                             {
00250                                 linkDestinationPoints[linkDestination.Key] =
      act.Geometry.Bounds.TopLeft.Transform(act.Transform);
00251                             }
00252                             else if (act.ActionType == RenderAction.ActionTypes.Text)
00253                             {
00254                                 linkDestinationPoints[linkDestination.Key] = new Avalonia.Point(0,
      0).Transform(act.Transform);
00255                             }
00256                             else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
00257                             {
00258                                 linkDestinationPoints[linkDestination.Key] =
      act.ImageDestination.Value.TopLeft.Transform(act.Transform);
00259                             }
00260 
00261                             return new RenderAction[] { act };
00262                         }));
00263 
00264                         if (url.StartsWith("#"))
00265                         {
00266                             if (!taggedActions.ContainsKey(url.Substring(1)))
00267                             {
00268                                 taggedActions.Add(url.Substring(1), (Func<RenderAction,
      IEnumerable<RenderAction>>)(act =>
00269                                 {
00270                                     if (act.ActionType == RenderAction.ActionTypes.Path)
00271                                     {
00272                                         linkDestinationPoints[url.Substring(1)] =
      act.Geometry.Bounds.TopLeft.Transform(act.Transform);
00273                                     }
00274                                     else if (act.ActionType == RenderAction.ActionTypes.Text)
00275                                     {
00276                                         linkDestinationPoints[url.Substring(1)] = new
      Avalonia.Point(0, 0).Transform(act.Transform);
00277                                     }
00278                                     else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
```

```
00279                                     {
00280                                         linkDestinationPoints[url.Substring(1)] =
      act.ImageDestination.Value.TopLeft.Transform(act.Transform);
00281                                     }
00282
00283                                     return new RenderAction[] { act };
00284                                 }));
00285                             }
00286                         }
00287                     }
00288
00289                     Control can = pag.PaintToCanvas(false, taggedActions, false,
      this.TextConversionOption);
00290
00291                     this.FindControl<ScrollViewer>("ScrollViewer").Content = can;
00292                     this.FindControl<ScrollViewer>("ScrollViewer").Padding = new Thickness(0, 0, 0,
      0);
00293                     lastRenderedWidth = width;
00294                     forcedRerender = false;
00295                 }
00296                 else
00297                 {
00298                     this.FindControl<ScrollViewer>("ScrollViewer").Padding = new Thickness(0, 0, width
      - lastRenderedWidth, 0);
00299                 }
00300             }
00301         }
00302
00303         private void InitializeComponent()
00304         {
00305             AvaloniaXamlLoader.Load(this);
00306         }
00307     }
00308 }
```

8.16 ImageUriParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using SixLabors.ImageSharp;
00019 using System;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023
00024 namespace VectSharp.ImageSharpUtils
00025 {
00026 /// <summary>
00027 /// Provides a method to parse an image URI into a page.
00028 /// </summary>
00029     public static class ImageURIParser
00030     {
00031 /// <summary>
00032 /// Parses an image URI into a page.  This is intended to replace the default image URI interpreter in
      <c>VectSharp.SVG.Parser.ParseImageURI</c>.  To do this, use something like:
00033 /// <code>VectSharp.SVG.Parser.ParseImageURI =
      VectSharp.ImageSharpUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00034 /// </summary>
00035 /// <param name="parseSVG">A function to parse an SVG image uri into a page.  You should pass
      <c>VectSharp.SVG.Parser.ParseSVGURI</c> as this argument.</param>
00036 /// <returns>A function to parse an image URI into a page.</returns>
00037         public static Func<string, bool, Page> Parser(Func<string, bool, Page> parseSVG)
00038         {
00039             return (string uri, bool interpolate) =>
00040             {
00041                 if (uri.StartsWith("data:"))
00042                 {
00043                     string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
      uri.IndexOf(":") - 1);
```

```
00044
00045                     string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") -
      uri.IndexOf(";") - 1);
00046
00047                     if (mimeType != "image/svg+xml")
00048                     {
00049                         int offset = uri.IndexOf(",") + 1;
00050
00051                         byte[] parsed;
00052
00053                         string substring = uri.Substring(offset);
00054
00055                         switch (type)
00056                         {
00057                             case "base64":
00058                                 parsed = Convert.FromBase64String(uri.Substring(offset));
00059                                 break;
00060                             case "":
00061                                 parsed = (from el in
      System.Web.HttpUtility.UrlDecode(uri.Substring(offset)) select (byte)el).ToArray();
00062                                 break;
00063                             default:
00064                                 throw new InvalidDataException("Unknown data stream type!");
00065                         }
00066
00067                         GCHandle handle = GCHandle.Alloc(parsed, GCHandleType.Pinned);
00068
00069                         RasterImageStream img = new RasterImageStream(handle.AddrOfPinnedObject(),
      parsed.Length, interpolate:  interpolate);
00070
00071                         handle.Free();
00072
00073                         Page pag = new Page(img.Width, img.Height);
00074
00075                         pag.Graphics.DrawRasterImage(0, 0, img);
00076
00077                         return pag;
00078                     }
00079                     else
00080                     {
00081                         return parseSVG(uri, interpolate);
00082                     }
00083                 }
00084
00085                 return null;
00086             };
00087         }
00088     }
00089 }
```

8.17 ImageURIParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023
00024 namespace VectSharp.MuPDFUtils
00025 {
00026 /// <summary>
00027 /// Provides a method to parse an image URI into a page.
00028 /// </summary>
00029     public static class ImageURIParser
00030     {
00031 /// <summary>
```

```
00032 /// Parses an image URI into a page.  This is intended to replace the default image URI interpreter in
      <c>VectSharp.SVG.Parser.ParseImageURI</c>.  To do this, use something like:
00033 /// <code>VectSharp.SVG.Parser.ParseImageURI =
      VectSharp.MuPDFUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00034 /// </summary>
00035 /// <param name="parseSVG">A function to parse an SVG image uri into a page.  You should pass
      <c>VectSharp.SVG.Parser.ParseSVGURI</c> as this argument.</param>
00036 /// <returns>A function to parse an image URI into a page.</returns>
00037         public static Func<string, bool, Page> Parser(Func<string, bool, Page> parseSVG)
00038         {
00039             return (string uri, bool interpolate) =>
00040             {
00041                 if (uri.StartsWith("data:"))
00042                 {
00043                     string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
      uri.IndexOf(":") - 1);
00044
00045                     string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") -
      uri.IndexOf(";") - 1);
00046
00047                     if (mimeType != "image/svg+xml")
00048                     {
00049                         int offset = uri.IndexOf(",") + 1;
00050
00051                         byte[] parsed;
00052
00053                         bool isVector = false;
00054
00055                         InputFileTypes fileType;
00056
00057                         switch (mimeType)
00058                         {
00059                             case "image/png":
00060                                 fileType = InputFileTypes.PNG;
00061                                 break;
00062                             case "image/jpeg":
00063                             case "image/jpg":
00064                                 fileType = InputFileTypes.JPEG;
00065                                 break;
00066                             case "image/gif":
00067                                 fileType = InputFileTypes.GIF;
00068                                 break;
00069                             case "image/bmp":
00070                                 fileType = InputFileTypes.BMP;
00071                                 break;
00072                             case "image/tiff":
00073                             case "image/tif":
00074                                 fileType = InputFileTypes.TIFF;
00075                                 break;
00076                             case "application/oxps":
00077                             case "application/vnd.ms-xpsdocument":
00078                                 fileType = InputFileTypes.XPS;
00079                                 isVector = true;
00080                                 break;
00081                             case "application/x-cbz":
00082                                 fileType = InputFileTypes.CBZ;
00083                                 break;
00084                             case "application/epub+zip":
00085                                 fileType = InputFileTypes.EPUB;
00086                                 isVector = true;
00087                                 break;
00088                             case "text/fb2+xml":
00089                                 fileType = InputFileTypes.FB2;
00090                                 break;
00091                             case "image/x-portable-anymap":
00092                                 fileType = InputFileTypes.PNM;
00093                                 break;
00094                             case "image/x-portable-arbitrarymap":
00095                                 fileType = InputFileTypes.PAM;
00096                                 break;
00097                             case "application/pdf":
00098                                 fileType = InputFileTypes.PDF;
00099                                 isVector = true;
00100                                 break;
00101                             default:
00102                                 fileType = InputFileTypes.PDF;
00103                                 break;
00104                         }
00105
00106                         string substring = uri.Substring(offset);
00107
00108
00109                         switch (type)
00110                         {
00111                             case "base64":
00112                                 parsed = Convert.FromBase64String(uri.Substring(offset));
00113                                 break;
```

```
00114                               case "":
00115                                   parsed = (from el in
      System.Web.HttpUtility.UrlDecode(uri.Substring(offset)) select (byte)el).ToArray();
00116                                   break;
00117                               default:
00118                                   throw new InvalidDataException("Unknown data stream type!");
00119                           }
00120
00121                           if (!isVector)
00122                           {
00123                               GCHandle handle = GCHandle.Alloc(parsed, GCHandleType.Pinned);
00124
00125                               RasterImageStream img = new RasterImageStream(handle.AddrOfPinnedObject(),
      parsed.Length, fileType, interpolate:  interpolate);
00126
00127                               handle.Free();
00128
00129                               Page pag = new Page(img.Width, img.Height);
00130
00131                               pag.Graphics.DrawRasterImage(0, 0, img);
00132
00133                               return pag;
00134                           }
00135                           else
00136                           {
00137                               string tempFile = Path.GetTempFileName();
00138
00139                               using (MuPDFContext context = new MuPDFContext())
00140                               {
00141                                   using (MuPDFDocument document = new MuPDFDocument(context, parsed,
      fileType))
00142                                   {
00143                                       MuPDFDocument.CreateDocument(context, tempFile,
      DocumentOutputFileTypes.SVG, true, document.Pages[0]);
00144                                   }
00145                               }
00146
00147                               string tbr = "data:image/svg+xml;," +
      System.Web.HttpUtility.UrlEncode(File.ReadAllText(tempFile));
00148
00149                               File.Delete(tempFile);
00150
00151                               return parseSVG(tbr, interpolate);
00152                           }
00153                       }
00154                       else
00155                       {
00156                           return parseSVG(uri, interpolate);
00157                       }
00158                   }
00159
00160                   return null;
00161               };
00162           }
00163       }
00164   }
```

8.18 RasterImages.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using SixLabors.ImageSharp;
00019 using SixLabors.ImageSharp.Advanced;
00020 using SixLabors.ImageSharp.Processing;
00021 using System;
00022 using System.IO;
00023 using System.Runtime.InteropServices;
00024
```

```
00025 namespace VectSharp.ImageSharpUtils
00026 {
00027 /// <summary>
00028 /// A <see cref="RasterImage"/> created from a file.
00029 /// </summary>
00030     public class RasterImageFile :  RasterImage
00031     {
00032 /// <summary>
00033 /// Creates a new <see cref="RasterImage"/> from the specified file.
00034 /// </summary>
00035 /// <param name="fileName">The path to the file containing the image.</param>
00036 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00037 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00038         public RasterImageFile(string fileName, bool alpha = true, bool interpolate = true)
00039         {
00040             Image image;
00041 
00042             if (alpha)
00043             {
00044                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(fileName);
00045             }
00046             else
00047             {
00048                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(fileName);
00049             }
00050 
00051             image.Mutate(x => x.AutoOrient());
00052 
00053             int stride = image.Width * (alpha ?  4 :  3);
00054             int size = stride * image.Height;
00055 
00056             IntPtr tbr = Marshal.AllocHGlobal(size);
00057             GC.AddMemoryPressure(size);
00058 
00059             IntPtr pointer = tbr;
00060 
00061             unsafe
00062             {
00063                 if (alpha)
00064                 {
00065                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00066 
00067                     for (int y = 0; y < image.Height; y++)
00068                     {
00069                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
00070 
00071                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
00072                         row.Span.CopyTo(newRow);
00073 
00074                         pointer = IntPtr.Add(pointer, stride);
00075                     }
00076                 }
00077                 else
00078                 {
00079                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00080 
00081                     for (int y = 0; y < image.Height; y++)
00082                     {
00083                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
      img.DangerousGetPixelRowMemory(y);
00084 
00085                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00086                         row.Span.CopyTo(newRow);
00087 
00088                         pointer = IntPtr.Add(pointer, stride);
00089                     }
00090                 }
00091             }
00092 
00093             this.Width = image.Width;
00094             this.Height = image.Height;
00095             this.HasAlpha = alpha;
00096             this.Interpolate = interpolate;
00097             this.Id = Guid.NewGuid().ToString();
00098             this.ImageDataAddress = tbr;
00099             this.DataHolder = new DisposableIntPtr(tbr);
00100         }
00101     }
00102 
00103 /// <summary>
```

```
00104 /// A <see cref="RasterImage"/> created from a stream.
00105 /// </summary>
00106     public class RasterImageStream :  RasterImage
00107     {
00108 /// <summary>
00109 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00110 /// </summary>
00111 /// <param name="imageStream">The stream containing the image data.</param>
00112 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00113 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00114         public RasterImageStream(Stream imageStream, bool alpha = true, bool interpolate = true)
00115         {
00116             Image image;
00117 
00118             if (alpha)
00119             {
00120                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(imageStream);
00121             }
00122             else
00123             {
00124                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(imageStream);
00125             }
00126 
00127             image.Mutate(x => x.AutoOrient());
00128 
00129             int stride = image.Width * (alpha ?  4 :  3);
00130             int size = stride * image.Height;
00131 
00132             IntPtr tbr = Marshal.AllocHGlobal(size);
00133             GC.AddMemoryPressure(size);
00134 
00135             IntPtr pointer = tbr;
00136 
00137             unsafe
00138             {
00139                 if (alpha)
00140                 {
00141                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00142 
00143                     for (int y = 0; y < image.Height; y++)
00144                     {
00145                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
00146 
00147                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
00148                         row.Span.CopyTo(newRow);
00149 
00150                         pointer = IntPtr.Add(pointer, stride);
00151                     }
00152                 }
00153                 else
00154                 {
00155                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00156 
00157                     for (int y = 0; y < image.Height; y++)
00158                     {
00159                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
      img.DangerousGetPixelRowMemory(y);
00160 
00161                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00162                         row.Span.CopyTo(newRow);
00163 
00164                         pointer = IntPtr.Add(pointer, stride);
00165                     }
00166                 }
00167             }
00168 
00169             this.Width = image.Width;
00170             this.Height = image.Height;
00171             this.HasAlpha = alpha;
00172             this.Interpolate = interpolate;
00173             this.Id = Guid.NewGuid().ToString();
00174             this.ImageDataAddress = tbr;
00175             this.DataHolder = new DisposableIntPtr(tbr);
00176         }
00177 
00178 /// <summary>
00179 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00180 /// </summary>
00181 /// <param name="imageAddress">A pointer to the address where the image data is contained.</param>
00182 /// <param name="imageLength">The length in bytes of the image data.</param>
```

```
00183 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00184 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00185         public RasterImageStream(IntPtr imageAddress, int imageLength, bool alpha = true, bool
      interpolate = true)
00186         {
00187             unsafe
00188             {
00189                 ReadOnlySpan<byte> imageSpan = new ReadOnlySpan<byte>((void*)imageAddress,
      imageLength);
00190
00191                 Image image;
00192
00193                 if (alpha)
00194                 {
00195                     image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(imageSpan);
00196                 }
00197                 else
00198                 {
00199                     image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(imageSpan);
00200                 }
00201
00202                 image.Mutate(x => x.AutoOrient());
00203
00204                 int stride = image.Width * (alpha ?  4 :  3);
00205                 int size = stride * image.Height;
00206
00207                 IntPtr tbr = Marshal.AllocHGlobal(size);
00208                 GC.AddMemoryPressure(size);
00209
00210                 IntPtr pointer = tbr;
00211
00212                 if (alpha)
00213                 {
00214                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00215
00216                     for (int y = 0; y < image.Height; y++)
00217                     {
00218                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
00219
00220                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
00221                         row.Span.CopyTo(newRow);
00222
00223                         pointer = IntPtr.Add(pointer, stride);
00224                     }
00225                 }
00226                 else
00227                 {
00228                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00229
00230                     for (int y = 0; y < image.Height; y++)
00231                     {
00232                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
      img.DangerousGetPixelRowMemory(y);
00233
00234                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00235                         row.Span.CopyTo(newRow);
00236
00237                         pointer = IntPtr.Add(pointer, stride);
00238                     }
00239                 }
00240
00241                 this.Width = image.Width;
00242                 this.Height = image.Height;
00243                 this.HasAlpha = alpha;
00244                 this.Interpolate = interpolate;
00245                 this.Id = Guid.NewGuid().ToString();
00246                 this.ImageDataAddress = tbr;
00247                 this.DataHolder = new DisposableIntPtr(tbr);
00248             }
00249         }
00250     }
00251 }
```

8.19 RasterImages.cs

```
00001 /*
```

```
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Runtime.InteropServices;
00022
00023 namespace VectSharp.MuPDFUtils
00024 {
00025 /// <summary>
00026 /// A <see cref="RasterImage"/> created from a file.
00027 /// </summary>
00028     public class RasterImageFile :  RasterImage
00029     {
00030 /// <summary>
00031 /// Creates a new <see cref="RasterImage"/> from the specified file.
00032 /// </summary>
00033 /// <param name="fileName">The path to the file containing the image.</param>
00034 /// <param name="pageNumber">The number of the page in the file from which the image should be
      created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00035 /// <param name="scale">The scale factor at which to render the image.</param>
00036 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00037 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00038         public RasterImageFile(string fileName, int pageNumber = 0, double scale = 1, bool alpha =
      true, bool interpolate = true)
00039         {
00040             using (MuPDFContext context = new MuPDFContext())
00041             {
00042                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
00043                 {
00044                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00045
00046                     this.Width = roundedBounds.Width;
00047                     this.Height = roundedBounds.Height;
00048
00049                     this.HasAlpha = alpha;
00050                     this.Interpolate = interpolate;
00051
00052                     this.Id = Guid.NewGuid().ToString();
00053
00054                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00055
00056                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00057                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00058                     GC.AddMemoryPressure(imageSize);
00059
00060                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00061                 }
00062             }
00063         }
00064     }
00065
00066 /// <summary>
00067 /// A <see cref="RasterImage"/> created from a stream.
00068 /// </summary>
00069     public class RasterImageStream :  RasterImage
00070     {
00071 /// <summary>
00072 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00073 /// </summary>
00074 /// <param name="imageStream">The stream containing the image data.</param>
00075 /// <param name="fileType">The type of the image contained in the stream.</param>
00076 /// <param name="pageNumber">The number of the page in the file from which the image should be
      created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00077 /// <param name="scale">The scale factor at which to render the image.</param>
00078 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00079 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
```

```
00080         public RasterImageStream(Stream imageStream, InputFileTypes fileType, int pageNumber = 0,
      double scale = 1, bool alpha = true, bool interpolate = true)
00081         {
00082             using (MuPDFContext context = new MuPDFContext())
00083             {
00084                 IntPtr originalImageAddress;
00085                 long originalImageLength;
00086 
00087                 IDisposable toBeDisposed = null;
00088                 GCHandle handleToFree;
00089 
00090                 if (imageStream is MemoryStream ms)
00091                 {
00092                     int origin = (int)ms.Seek(0, SeekOrigin.Begin);
00093                     originalImageLength = ms.Length;
00094 
00095                     handleToFree = GCHandle.Alloc(ms.GetBuffer(), GCHandleType.Pinned);
00096                     originalImageAddress = handleToFree.AddrOfPinnedObject();
00097                 }
00098                 else
00099                 {
00100                     MemoryStream mem = new MemoryStream((int)imageStream.Length);
00101                     imageStream.CopyTo(mem);
00102 
00103                     toBeDisposed = mem;
00104 
00105                     int origin = (int)mem.Seek(0, SeekOrigin.Begin);
00106                     originalImageLength = mem.Length;
00107 
00108                     handleToFree = GCHandle.Alloc(mem.GetBuffer(), GCHandleType.Pinned);
00109                     originalImageAddress = handleToFree.AddrOfPinnedObject();
00110                 }
00111 
00112                 using (MuPDFDocument document = new MuPDFDocument(context, originalImageAddress,
      originalImageLength, fileType))
00113                 {
00114                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00115 
00116                     this.Width = roundedBounds.Width;
00117                     this.Height = roundedBounds.Height;
00118 
00119                     this.HasAlpha = alpha;
00120                     this.Interpolate = interpolate;
00121 
00122                     this.Id = Guid.NewGuid().ToString();
00123 
00124                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00125 
00126                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00127                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00128                     GC.AddMemoryPressure(imageSize);
00129 
00130                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00131                 }
00132 
00133                 handleToFree.Free();
00134                 toBeDisposed?.Dispose();
00135             }
00136         }
00137 
00138 /// <summary>
00139 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00140 /// </summary>
00141 /// <param name="imageAddress">A pointer to the address where the image data is contained.</param>
00142 /// <param name="imageLength">The length in bytes of the image data.</param>
00143 /// <param name="fileType">The type of the image contained in the stream.</param>
00144 /// <param name="pageNumber">The number of the page in the file from which the image should be
      created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00145 /// <param name="scale">The scale factor at which to render the image.</param>
00146 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00147 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00148         public RasterImageStream(IntPtr imageAddress, long imageLength, InputFileTypes fileType, int
      pageNumber = 0, double scale = 1, bool alpha = true, bool interpolate = true)
00149         {
00150             using (MuPDFContext context = new MuPDFContext())
00151             {
00152                 using (MuPDFDocument document = new MuPDFDocument(context, imageAddress, imageLength,
      fileType))
00153                 {
00154                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00155 
00156                     this.Width = roundedBounds.Width;
00157                     this.Height = roundedBounds.Height;
```

```
00158
00159                     this.HasAlpha = alpha;
00160                     this.Interpolate = interpolate;
00161
00162                     this.Id = Guid.NewGuid().ToString();
00163
00164                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00165
00166                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00167                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00168                     GC.AddMemoryPressure(imageSize);
00169
00170                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00171                 }
00172             }
00173         }
00174     }
00175 }
```

8.20 PDFContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.ComponentModel;
00022 using System.IO;
00023 using System.IO.Compression;
00024 using System.Linq;
00025 using System.Runtime.InteropServices;
00026 using System.Text;
00027 using VectSharp.Filters;
00028
00029 namespace VectSharp.PDF
00030 {
00031     internal enum SegmentType
00032     {
00033         Move, Line, CubicBezier, Close
00034     }
00035
00036     internal abstract class Segment
00037     {
00038         public abstract SegmentType Type { get; }
00039         public Point[] Points { get; protected set; }
00040
00041         public virtual Point Point
00042         {
00043             get
00044             {
00045                 return Points[Points.Length - 1];
00046             }
00047         }
00048
00049         public abstract Segment Clone();
00050     }
00051
00052     internal class MoveSegment :  Segment
00053     {
00054         public override SegmentType Type => SegmentType.Move;
00055
00056         public MoveSegment(Point p)
00057         {
00058             this.Points = new Point[] { p };
00059         }
00060
```

```
00061         public MoveSegment(double x, double y)
00062         {
00063             this.Points = new Point[] { new Point(x, y) };
00064         }
00065 
00066         public override Segment Clone()
00067         {
00068             return new MoveSegment(this.Point);
00069         }
00070     }
00071 
00072     internal class LineSegment :  Segment
00073     {
00074         public override SegmentType Type => SegmentType.Line;
00075 
00076         public LineSegment(Point p)
00077         {
00078             this.Points = new Point[] { p };
00079         }
00080 
00081         public LineSegment(double x, double y)
00082         {
00083             this.Points = new Point[] { new Point(x, y) };
00084         }
00085 
00086         public override Segment Clone()
00087         {
00088             return new LineSegment(this.Point);
00089         }
00090     }
00091 
00092     internal class CloseSegment :  Segment
00093     {
00094         public override SegmentType Type => SegmentType.Close;
00095 
00096         public CloseSegment() { }
00097 
00098         public override Segment Clone()
00099         {
00100             return new CloseSegment();
00101         }
00102     }
00103 
00104     internal class CubicBezierSegment :  Segment
00105     {
00106         public override SegmentType Type => SegmentType.CubicBezier;
00107         public CubicBezierSegment(double x1, double y1, double x2, double y2, double x3, double y3)
00108         {
00109             Points = new Point[] { new Point(x1, y1), new Point(x2, y2), new Point(x3, y3) };
00110         }
00111 
00112         public CubicBezierSegment(Point p1, Point p2, Point p3)
00113         {
00114             Points = new Point[] { p1, p2, p3 };
00115         }
00116 
00117         public override Segment Clone()
00118         {
00119             return new CubicBezierSegment(Points[0], Points[1], Points[2]);
00120         }
00121     }
00122 
00123     internal interface IFigure
00124     {
00125         Brush Fill { get; }
00126         Brush Stroke { get; }
00127         double LineWidth { get; }
00128         LineCaps LineCap { get; }
00129         LineJoins LineJoin { get; }
00130         LineDash LineDash { get; }
00131         bool IsClipping { get; }
00132         string Tag { get; }
00133         Rectangle GetBounds();
00134     }
00135 
00136     internal class TransformFigure :  IFigure
00137     {
00138         public enum TransformTypes
00139         {
00140             Transform, Save, Restore
00141         }
00142 
00143         public TransformTypes TransformType { get; }
00144 
00145         public Brush Fill { get; }
00146         public Brush Stroke { get; }
00147 
```

```
00148         public bool IsClipping { get; }
00149         public double LineWidth { get; }
00150 
00151         public double[,] TransformationMatrix { get; }
00152 
00153         public LineCaps LineCap { get; }
00154 
00155         public LineJoins LineJoin { get; }
00156 
00157         public LineDash LineDash { get; }
00158         public string Tag { get; }
00159 
00160         public TransformFigure(TransformTypes type, double[,] transformationMatrix, string tag)
00161         {
00162             this.TransformType = type;
00163             this.TransformationMatrix = transformationMatrix;
00164             this.Tag = tag;
00165         }
00166 
00167         public Rectangle GetBounds()
00168         {
00169             return Rectangle.NaN;
00170         }
00171     }
00172 
00173     internal class RasterImageFigure :  IFigure
00174     {
00175 
00176         public Brush Fill { get; }
00177         public Brush Stroke { get; }
00178 
00179         public bool IsClipping { get; }
00180         public double LineWidth { get; }
00181 
00182         public double[,] TransformationMatrix { get; }
00183 
00184         public LineCaps LineCap { get; }
00185 
00186         public LineJoins LineJoin { get; }
00187 
00188         public LineDash LineDash { get; }
00189 
00190         public RasterImage Image { get; }
00191 
00192         public string Tag { get; }
00193 
00194         public RasterImageFigure(RasterImage image, string tag)
00195         {
00196             this.Image = image;
00197             this.Tag = tag;
00198         }
00199 
00200         public Rectangle GetBounds()
00201         {
00202             return new Rectangle(0, 0, 1, 1);
00203         }
00204     }
00205 
00206     internal class PathFigure :  IFigure
00207     {
00208         public Brush Fill { get; }
00209         public FillRule FillRule { get; }
00210         public Brush Stroke { get; }
00211         public bool IsClipping { get; }
00212         public double LineWidth { get; }
00213 
00214         public LineCaps LineCap { get; }
00215 
00216         public LineJoins LineJoin { get; }
00217 
00218         public LineDash LineDash { get; }
00219         public Segment[] Segments { get; }
00220 
00221         public string Tag { get; }
00222 
00223         private Rectangle Bounds { get; }
00224 
00225         public PathFigure(IEnumerable<Segment> segments, Rectangle bounds, Brush fill, Brush stroke,
      double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, bool isClipping, FillRule
      fillRule, string tag)
00226         {
00227             List<Segment> segs = new List<Segment>();
00228 
00229             foreach (Segment s in segments)
00230             {
00231                 segs.Add(s.Clone());
00232             }
```

```
00233
00234             this.Segments = segs.ToArray();
00235
00236             Fill = fill;
00237             Stroke = stroke;
00238             LineWidth = lineWidth;
00239             LineCap = lineCap;
00240             LineJoin = lineJoin;
00241             LineDash = lineDash;
00242             IsClipping = isClipping;
00243             this.Tag = tag;
00244             this.FillRule = fillRule;
00245
00246             if (stroke == null)
00247             {
00248                 this.Bounds = bounds;
00249             }
00250             else
00251             {
00252                 this.Bounds = new Rectangle(bounds.Location.X - lineWidth * 0.5, bounds.Location.Y -
      lineWidth * 0.5, bounds.Size.Width + lineWidth, bounds.Size.Height + lineWidth);
00253             }
00254         }
00255
00256         public Rectangle GetBounds()
00257         {
00258             return Bounds;
00259         }
00260     }
00261
00262     internal class TextFigure :  IFigure
00263     {
00264         public Brush Fill { get; }
00265         public Brush Stroke { get; }
00266         public double LineWidth { get; }
00267         public bool IsClipping { get; }
00268         public LineCaps LineCap { get; }
00269
00270         public LineJoins LineJoin { get; }
00271
00272         public LineDash LineDash { get; }
00273
00274         public string Text { get; }
00275
00276         public Font Font { get; }
00277
00278         public Point Position { get; }
00279
00280         public TextBaselines TextBaseline { get; }
00281
00282         public string Tag { get; }
00283
00284         public TextFigure(string text, Font font, Point position, TextBaselines textBaseline, Brush
      fill, Brush stroke, double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string
      tag)
00285         {
00286             Text = text;
00287             Font = font;
00288             Position = position;
00289             TextBaseline = textBaseline;
00290
00291             Fill = fill;
00292             Stroke = stroke;
00293             LineWidth = lineWidth;
00294             LineCap = lineCap;
00295             LineJoin = lineJoin;
00296             LineDash = lineDash;
00297             this.Tag = tag;
00298         }
00299
00300         public Rectangle GetBounds()
00301         {
00302             Font.DetailedFontMetrics metrics = this.Font.MeasureTextAdvanced(this.Text);
00303
00304             switch (this.TextBaseline)
00305             {
00306                 case TextBaselines.Top:
00307                     return new Rectangle(this.Position, new Size(metrics.Width, metrics.Height));
00308                 case TextBaselines.Bottom:
00309                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Height,
      metrics.Width, metrics.Height);
00310                 case TextBaselines.Middle:
00311                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Height * 0.5,
      metrics.Width, metrics.Height);
00312                 case TextBaselines.Baseline:
00313                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Top,
      metrics.Width, metrics.Height);
```

```
00314                 default:
00315                     throw new System.ArgumentOutOfRangeException(nameof(TextBaseline),
    this.TextBaseline, "Invalid text baseline!");
00316             }
00317         }
00318     }
00319
00320     internal class PDFFontDescriptor
00321     {
00322         private static readonly Random TagGenerator = new Random();
00323         private static readonly List<string> TagCache = new List<string>();
00324
00325         public string FontName { get; }
00326         public string FontFamily { get; }
00327         public uint Flags { get; }
00328         public double[] FontBBox { get; }
00329         public int ItalicAngle => 0;
00330         public double Ascent { get; }
00331         public double Descent { get; }
00332         public double CapHeight { get { return Ascent; } }
00333         public int StemV => 80;
00334         public int StemH => 80;
00335
00336         public PDFFontDescriptor(TrueTypeFile ttf, bool isSubset, bool isSymbolic)
00337         {
00338             this.Ascent = ttf.Get1000EmAscent();
00339             this.Descent = ttf.Get1000EmDescent();
00340
00341             this.FontBBox = new double[] { ttf.Get1000EmXMin(), ttf.Get1000EmYMin(),
    ttf.Get1000EmXMax(), ttf.Get1000EmYMax() };
00342
00343             bool fixedPitch = ttf.IsFixedPitch();
00344
00345             bool serif = ttf.IsSerif();
00346
00347             bool script = ttf.IsScript();
00348
00349             bool italic = ttf.IsBold();
00350
00351             bool allCap = false;
00352
00353             bool smallCap = false;
00354
00355             bool forceBold = false;
00356
00357             this.Flags = (fixedPitch ? 1U : 0) | (serif ? 1U << 1 : 0) | (isSymbolic ? 1U << 2 : 0)
    | (script ? 1U << 3 : 0) | (!isSymbolic ? 1U << 5 : 0) | (italic ? 1U << 6 : 0) | (allCap ? 1U <<
    16 : 0) | (smallCap ? 1U << 17 : 0) | (forceBold ? 1U << 18 : 0);
00358
00359             this.FontName = ttf.GetFontName();
00360
00361             this.FontFamily = ttf.GetFontFamilyName();
00362
00363             if (string.IsNullOrEmpty(this.FontFamily))
00364             {
00365                 this.FontFamily = FontName;
00366             }
00367
00368             if (isSubset)
00369             {
00370                 string randString = "";
00371
00372                 while (randString.Length == 0 || TagCache.Contains(randString))
00373                 {
00374                     randString = "";
00375                     for (int i = 0; i < 6; i++)
00376                     {
00377                         randString += (char)TagGenerator.Next(65, 91);
00378                     }
00379                 }
00380
00381                 this.FontName = randString + "+" + this.FontName;
00382             }
00383         }
00384     }
00385
00386     internal class PDFContext :  IGraphicsContext
00387     {
00388         public string Tag { get; set; }
00389         public double Width { get; }
00390         public double Height { get; }
00391
00392
00393         private List<Segment> _currentFigure;
00394
00395         internal List<IFigure> _figures;
00396
```

```
00397         private Brush _strokeStyle;
00398         private Brush _fillStyle;
00399         private LineDash _lineDash;
00400 
00401         private readonly bool _textToPaths;
00402 
00403         private PDFContextInterpreter.FilterOption _filterOption;
00404 
00405         public PDFContext(double width, double height, Colour background, bool textToPaths,
      PDFContextInterpreter.FilterOption filterOption)
00406         {
00407             this.Width = width;
00408             this.Height = height;
00409 
00410             _currentFigure = new List<Segment>();
00411             _figures = new List<IFigure>();
00412 
00413             _strokeStyle = Colour.FromRgb(0, 0, 0);
00414             _fillStyle = Colour.FromRgb(0, 0, 0);
00415             LineWidth = 1;
00416 
00417             LineCap = LineCaps.Butt;
00418             LineJoin = LineJoins.Miter;
00419             _lineDash = new LineDash(0, 0, 0);
00420 
00421             _textToPaths = textToPaths;
00422 
00423             Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica),
      12);
00424 
00425             TextBaseline = TextBaselines.Top;
00426 
00427             this.Translate(0, height);
00428             this.Scale(1, -1);
00429 
00430             this.Rectangle(0, 0, width, height);
00431             this.SetFillStyle(background);
00432             this.Fill(FillRule.NonZeroWinding);
00433 
00434             this.SetFillStyle(Colour.FromRgb(0, 0, 0));
00435 
00436             this._filterOption = filterOption;
00437         }
00438 
00439 
00440         public void MoveTo(double x, double y)
00441         {
00442             _currentFigure.Add(new MoveSegment(x, y));
00443         }
00444 
00445         public void LineTo(double x, double y)
00446         {
00447             _currentFigure.Add(new LineSegment(x, y));
00448         }
00449 
00450         public void Close()
00451         {
00452             _currentFigure.Add(new CloseSegment());
00453         }
00454 
00455         public void Rectangle(double x0, double y0, double width, double height)
00456         {
00457             MoveTo(x0, y0);
00458             LineTo(x0 + width, y0);
00459             LineTo(x0 + width, y0 + height);
00460             LineTo(x0, y0 + height);
00461             Close();
00462         }
00463         public void SetStrokeStyle((int r, int g, int b, double a) style)
00464         {
00465             _strokeStyle = Colour.FromRgba(style.r, style.g, style.b, style.a);
00466         }
00467 
00468         public void SetStrokeStyle(Brush style)
00469         {
00470             _strokeStyle = style;
00471         }
00472 
00473         public void SetFillStyle((int r, int g, int b, double a) style)
00474         {
00475             _fillStyle = Colour.FromRgba(style.r, style.g, style.b, style.a);
00476         }
00477 
00478         public void SetFillStyle(Brush style)
00479         {
00480             _fillStyle = style;
00481         }
```

```
00482
00483
00484         public Brush FillStyle { get { return _fillStyle; } }
00485         public Brush StrokeStyle { get { return _strokeStyle; } }
00486
00487         public double LineWidth { get; set; }
00488
00489         public LineCaps LineCap { get; set; }
00490         public LineJoins LineJoin { get; set; }
00491
00492         internal static bool IsCompatible(Brush brush)
00493         {
00494             if (brush is SolidColourBrush)
00495             {
00496                 return true;
00497             }
00498             else if (brush is GradientBrush gradient)
00499             {
00500                 foreach (GradientStop stop in gradient.GradientStops)
00501                 {
00502                     if (stop.Colour.A != 1)
00503                     {
00504                         return false;
00505                     }
00506                 }
00507                 return true;
00508             }
00509             else
00510             {
00511                 return false;
00512             }
00513         }
00514
00515         private static Brush RemoveAlpha(Brush brush)
00516         {
00517             if (brush is SolidColourBrush)
00518             {
00519                 return brush;
00520             }
00521             else if (brush is LinearGradientBrush linear)
00522             {
00523                 return new LinearGradientBrush(linear.StartPoint, linear.EndPoint, from el in
      linear.GradientStops select new GradientStop(el.Colour.WithAlpha(1.0), el.Offset));
00524             }
00525             else if (brush is RadialGradientBrush radial)
00526             {
00527                 return new RadialGradientBrush(radial.FocalPoint, radial.Centre, radial.Radius, from
      el in radial.GradientStops select new GradientStop(el.Colour.WithAlpha(1.0), el.Offset));
00528             }
00529             else
00530             {
00531                 return null;
00532             }
00533         }
00534
00535         internal static Brush GetAlphaBrush(Brush brush)
00536         {
00537             if (brush is SolidColourBrush)
00538             {
00539                 return brush;
00540             }
00541             else if (brush is LinearGradientBrush linear)
00542             {
00543                 return new LinearGradientBrush(linear.StartPoint, linear.EndPoint, from el in
      linear.GradientStops select new GradientStop(Colour.FromRgb(el.Colour.A, el.Colour.A, el.Colour.A),
      el.Offset));
00544             }
00545             else if (brush is RadialGradientBrush radial)
00546             {
00547                 return new RadialGradientBrush(radial.FocalPoint, radial.Centre, radial.Radius, from
      el in radial.GradientStops select new GradientStop(Colour.FromRgb(el.Colour.A, el.Colour.A,
      el.Colour.A), el.Offset));
00548             }
00549             else
00550             {
00551                 return null;
00552             }
00553         }
00554
00555         private static GraphicsPath GetGraphicsPath(IEnumerable<Segment> segments)
00556         {
00557             GraphicsPath tbr = new GraphicsPath();
00558
00559             foreach (Segment seg in segments)
00560             {
00561                 switch (seg.Type)
00562                 {
```

```
00563                     case SegmentType.Close:
00564                         tbr.Close();
00565                         break;
00566                     case SegmentType.Move:
00567                         tbr.MoveTo(seg.Point);
00568                         break;
00569                     case SegmentType.Line:
00570                         tbr.LineTo(seg.Point);
00571                         break;
00572                     case SegmentType.CubicBezier:
00573                         tbr.CubicBezierTo(seg.Points[0], seg.Points[1], seg.Points[2]);
00574                         break;
00575                 }
00576             }
00577 
00578             return tbr;
00579         }
00580 
00581         private Graphics GetCurrentFigureMask(Brush brush, bool stroke, bool blackBackground = false)
00582         {
00583             Graphics gpr = new Graphics();
00584 
00585             GraphicsPath path = GetGraphicsPath(_currentFigure);
00586 
00587             if (!stroke)
00588             {
00589                 gpr.FillPath(path, brush);
00590             }
00591             else
00592             {
00593                 gpr.StrokePath(path, brush, LineWidth, LineCap, LineJoin, _lineDash);
00594             }
00595 
00596             if (blackBackground)
00597             {
00598                 Rectangle bounds = gpr.GetBounds();
00599 
00600                 Graphics gpr2 = new Graphics();
00601 
00602                 gpr2.FillRectangle(bounds.Location.X, bounds.Location.Y, bounds.Size.Width,
      bounds.Size.Height, Colours.White);
00603                 gpr2.DrawGraphics(0, 0, gpr);
00604 
00605                 gpr = gpr2;
00606             }
00607 
00608             return gpr;
00609         }
00610 
00611         public void Fill(FillRule fillRule)
00612         {
00613             if (IsCompatible(_fillStyle))
00614             {
00615                 _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, _fillStyle, null,
      0, LineCaps.Butt, LineJoins.Bevel, new LineDash(0, 0, 0), false, fillRule, this.Tag));
00616             }
00617             else
00618             {
00619                 _figures.Add(new PathFigure(_currentFigure,
      GetGraphicsPath(_currentFigure).GetBounds(), _fillStyle, null, 0, LineCaps.Butt, LineJoins.Bevel, new
      LineDash(0, 0, 0), false, fillRule, this.Tag));
00620             }
00621 
00622             _currentFigure = new List<Segment>();
00623         }
00624 
00625         public void Stroke()
00626         {
00627             if (IsCompatible(_strokeStyle))
00628             {
00629                 _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, null,
      _strokeStyle, LineWidth, LineCap, LineJoin, _lineDash, false, FillRule.NonZeroWinding, this.Tag));
00630             }
00631             else
00632             {
00633                 _figures.Add(new PathFigure(_currentFigure,
      GetGraphicsPath(_currentFigure).GetBounds(), null, _strokeStyle, LineWidth, LineCap, LineJoin,
      _lineDash, false, FillRule.NonZeroWinding, this.Tag));
00634             }
00635 
00636             _currentFigure = new List<Segment>();
00637         }
00638 
00639         public void SetClippingPath()
00640         {
00641             _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, null, null, 0,
      LineCaps.Butt, LineJoins.Bevel, new LineDash(0, 0, 0), true, FillRule.NonZeroWinding, this.Tag));
```

```
00642             _currentFigure = new List<Segment>();
00643         }
00644
00645         public void CubicBezierTo(double x1, double y1, double x2, double y2, double x3, double y3)
00646         {
00647             _currentFigure.Add(new CubicBezierSegment(x1, y1, x2, y2, x3, y3));
00648         }
00649
00650         public void SetLineDash(LineDash dash)
00651         {
00652             _lineDash = dash;
00653         }
00654
00655         public Font Font { get; set; }
00656
00657         private void PathText(string text, double x, double y)
00658         {
00659             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00660
00661             for (int j = 0; j < textPath.Segments.Count; j++)
00662             {
00663                 switch (textPath.Segments[j].Type)
00664                 {
00665                     case VectSharp.SegmentType.Move:
00666                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00667                         break;
00668                     case VectSharp.SegmentType.Line:
00669                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00670                         break;
00671                     case VectSharp.SegmentType.CubicBezier:
00672                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
    textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
    textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00673                         break;
00674                     case VectSharp.SegmentType.Close:
00675                         this.Close();
00676                         break;
00677                 }
00678             }
00679         }
00680
00681         public void FillText(string text, double x, double y)
00682         {
00683             if (!_textToPaths)
00684             {
00685                 _figures.Add(new TextFigure(text, Font, new Point(x, y), TextBaseline, _fillStyle,
    null, 0, LineCaps.Butt, LineJoins.Miter, new LineDash(0, 0, 0), this.Tag));
00686             }
00687             else
00688             {
00689                 PathText(text, x, y);
00690                 Fill(FillRule.NonZeroWinding);
00691             }
00692         }
00693
00694         public TextBaselines TextBaseline { get; set; }
00695
00696         public void Restore()
00697         {
00698             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Restore, null, this.Tag));
00699         }
00700
00701         public void Rotate(double angle)
00702         {
00703             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
    { Math.Cos(angle), Math.Sin(angle), 0 }, { -Math.Sin(angle), Math.Cos(angle), 0 }, { 0, 0, 1 } },
    this.Tag));
00704         }
00705
00706         public void Save()
00707         {
00708             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Save, null, this.Tag));
00709         }
00710
00711
00712         public void StrokeText(string text, double x, double y)
00713         {
00714             if (!_textToPaths)
00715             {
00716                 _figures.Add(new TextFigure(text, Font, new Point(x, y), TextBaseline, null,
    _strokeStyle, LineWidth, LineCap, LineJoin, _lineDash, this.Tag));
00717             }
00718             else
00719             {
00720                 PathText(text, x, y);
00721                 Stroke();
00722             }
```

```
00723         }
00724
00725         public void Translate(double x, double y)
00726         {
00727             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { 1, 0, x }, { 0, 1, y }, { 0, 0, 1 } }, this.Tag));
00728         }
00729
00730         public void Scale(double scaleX, double scaleY)
00731         {
00732             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { scaleX, 0, 0 }, { 0, scaleY, 0 }, { 0, 0, 1 } }, this.Tag));
00733         }
00734
00735         public void Transform(double a, double b, double c, double d, double e, double f)
00736         {
00737             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { a, b, e }, { c, d, f }, { 0, 0, 1 } }, this.Tag));
00738         }
00739
00740         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00741         {
00742             Save();
00743
00744             MoveTo(destinationX, destinationY);
00745             LineTo(destinationX + destinationWidth, destinationY);
00746             LineTo(destinationX + destinationWidth, destinationY + destinationHeight);
00747             LineTo(destinationX, destinationY + destinationHeight);
00748             Close();
00749             SetClippingPath();
00750
00751             double sourceRectX = (double)sourceX / image.Width;
00752             double sourceRectY = 1 - (double)sourceY / image.Height;
00753             double sourceRectWidth = (double)sourceWidth / image.Width;
00754             double sourceRectHeight = -(double)sourceHeight / image.Height;
00755
00756             double scaleX = destinationWidth / sourceRectWidth;
00757             double scaleY = destinationHeight / sourceRectHeight;
00758
00759             double translationX = destinationX / scaleX - sourceRectX;
00760             double translationY = destinationY / scaleY - sourceRectY;
00761
00762
00763             Scale(scaleX, scaleY);
00764             Translate(translationX, translationY);
00765
00766             _figures.Add(new RasterImageFigure(image, this.Tag));
00767
00768             Restore();
00769         }
00770
00771         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00772         {
00773             if (this._filterOption.Operation ==
      PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll)
00774             {
00775                 double scale = this._filterOption.RasterisationResolution;
00776
00777                 Rectangle bounds = graphics.GetBounds();
00778
00779                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00780
00781                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00782                 {
00783                     if (!this._filterOption.RasterisationResolutionRelative)
00784                     {
00785                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
00786                     }
00787
00788                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
00789                     {
00790                         RasterImage filtered = null;
00791
00792                         if (filter is ILocationInvariantFilter locInvFilter)
00793                         {
00794                             filtered = locInvFilter.Filter(rasterised, scale);
00795                         }
00796                         else if (filter is IFilterWithLocation filterWithLoc)
00797                         {
00798                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00799                         }
00800
00801                         if (filtered != null)
```

```
00802                         {
00803                             rasterised.Dispose();
00804 
00805                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00806                         }
00807                     }
00808                     else
00809                     {
00810                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
00811 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
00812 • Provide your own implementation of Graphics.RasterisationMethod.
00813 • Set the FilterOption.Operation to ""IgnoreAll"" or ""SkipAll"".");
00814                     }
00815                 }
00816             }
00817             else if (this._filterOption.Operation ==
      PDFContextInterpreter.FilterOption.FilterOperations.IgnoreAll)
00818             {
00819                 graphics.CopyToIGraphicsContext(this);
00820             }
00821             else
00822             {
00823 
00824             }
00825         }
00826     }
00827 
00828 /// <summary>
00829 /// Contains methods to render a <see cref="Document"/> as a PDF document.
00830 /// </summary>
00831     public static class PDFContextInterpreter
00832     {
00833         private static string GetKernedString(string text, Font font)
00834         {
00835             List<(string, Point)> tSpans = new List<(string, Point)>();
00836 
00837             StringBuilder currentRun = new StringBuilder();
00838             Point currentKerning = new Point();
00839 
00840             Point currentGlyphPlacementDelta = new Point();
00841             Point currentGlyphAdvanceDelta = new Point();
00842             Point nextGlyphPlacementDelta = new Point();
00843             Point nextGlyphAdvanceDelta = new Point();
00844 
00845             for (int i = 0; i < text.Length; i++)
00846             {
00847                 if (i < text.Length - 1)
00848                 {
00849                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00850                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00851                     nextGlyphAdvanceDelta = new Point();
00852                     nextGlyphPlacementDelta = new Point();
00853 
00854                     TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00855 
00856                     if (kerning != null)
00857                     {
00858                         currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00859                         currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00860 
00861                         nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00862                         nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00863                     }
00864                 }
00865 
00866                 if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00867                 {
00868                     if (currentRun.Length > 0)
00869                     {
00870                         tSpans.Add((currentRun.ToString(), currentKerning));
00871 
00872                         tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X,
      currentGlyphPlacementDelta.Y)));
00873 
00874                         currentRun.Clear();
00875                         currentKerning = new Point(currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X, currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y);
00876                     }
```

```
00877                     else
00878                     {
00879                         tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X +
      currentKerning.X, currentGlyphPlacementDelta.Y + currentKerning.Y)));
00880 
00881                         currentRun.Clear();
00882                         currentKerning = new Point(currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X, currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y);
00883                     }
00884                 }
00885                 else
00886                 {
00887                     currentRun.Append(text[i]);
00888                 }
00889             }
00890 
00891             if (currentRun.Length > 0)
00892             {
00893                 tSpans.Add((currentRun.ToString(), currentKerning));
00894             }
00895 
00896             StringBuilder sb = new StringBuilder();
00897             sb.Append("[");
00898 
00899             for (int i = 0; i < tSpans.Count; i++)
00900             {
00901                 if (tSpans[i].Item2.X != 0)
00902                 {
00903                     sb.Append((-tSpans[i].Item2.X).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture));
00904                 }
00905 
00906                 sb.Append("(");
00907                 sb.Append(EscapeStringForPDF(tSpans[i].Item1));
00908                 sb.Append(")");
00909             }
00910 
00911             sb.Append("]");
00912 
00913             return sb.ToString();
00914         }
00915 
00916 
00917         private static string EscapeStringForPDF(string str)
00918         {
00919             StringBuilder sb = new StringBuilder();
00920 
00921             for (int i = 0; i < str.Length; i++)
00922             {
00923                 char ch = str[i];
00924 
00925                 if (CP1252Chars.Contains(ch))
00926                 {
00927                     if ((int)ch < 128)
00928                     {
00929                         if (!"\n\r\t\b\f()\\".Contains(ch))
00930                         {
00931                             sb.Append(ch);
00932                         }
00933                         else
00934                         {
00935                             switch (ch)
00936                             {
00937                                 case '\n':
00938                                     sb.Append("\\n");
00939                                     break;
00940                                 case '\r':
00941                                     sb.Append("\\r");
00942                                     break;
00943                                 case '\t':
00944                                     sb.Append("\\t");
00945                                     break;
00946                                 case '\b':
00947                                     sb.Append("\\b");
00948                                     break;
00949                                 case '\f':
00950                                     sb.Append("\\f");
00951                                     break;
00952                                 case '\\':
00953                                     sb.Append("\\\\");
00954                                     break;
00955                                 case '(':
00956                                     sb.Append("\\(");
00957                                     break;
00958                                 case ')':
00959                                     sb.Append("\\)");
00960                                     break;
```

```
00961                               }
00962                           }
00963                       }
00964                       else
00965                       {
00966                           string octal = Convert.ToString((int)ch, 8);
00967                           while (octal.Length < 3)
00968                           {
00969                               octal = "0" + octal;
00970                           }
00971                           sb.Append("\\" + octal);
00972                       }
00973                   }
00974                   else
00975                   {
00976                       sb.Append('?');
00977                   }
00978               }
00979               return sb.ToString();
00980           }
00981
00982
00983           private static string EscapeSymbolStringForPDF(string str, Dictionary<char, int> glyphIndices)
00984           {
00985               StringBuilder sb = new StringBuilder();
00986
00987               for (int i = 0; i < str.Length; i++)
00988               {
00989                   sb.Append((glyphIndices[str[i]]).ToString("X4"));
00990               }
00991               return sb.ToString();
00992           }
00993
00994           private static Dictionary<string, FontFamily> GetFontFamilies(PDFContext[] pdfContexts)
00995           {
00996               Dictionary<string, FontFamily> tbr = new Dictionary<string, FontFamily>();
00997
00998               foreach (PDFContext ctx in pdfContexts)
00999               {
01000                   foreach (IFigure act in ctx._figures)
01001                   {
01002                       if (act is TextFigure figure &&
      !tbr.ContainsKey(figure.Font.FontFamily.FamilyName))
01003                       {
01004                           tbr.Add(figure.Font.FontFamily.FamilyName,
      FontFamily.ResolveFontFamily(figure.Font.FontFamily.FamilyName));
01005                       }
01006                   }
01007               }
01008
01009               return tbr;
01010           }
01011
01012           private static Dictionary<string, HashSet<char» GetUsedChars(PDFContext[] pdfContexts)
01013           {
01014               Dictionary<string, HashSet<char» tbr = new Dictionary<string, HashSet<char»();
01015
01016               foreach (PDFContext ctx in pdfContexts)
01017               {
01018                   foreach (IFigure act in ctx._figures)
01019                   {
01020                       if (act is TextFigure figure &&
      !tbr.ContainsKey(figure.Font.FontFamily.FamilyName))
01021                       {
01022                           tbr.Add(figure.Font.FontFamily.FamilyName, new HashSet<char>(figure.Text));
01023                       }
01024                       else if (act is TextFigure figure1)
01025                       {
01026                           string txt = figure1.Text;
01027                           for (int i = 0; i < txt.Length; i++)
01028                           {
01029                               tbr[figure1.Font.FontFamily.FamilyName].Add(txt[i]);
01030                           }
01031                       }
01032                   }
01033               }
01034
01035               return tbr;
01036           }
01037
01038
01039           private static double[] GetAlphas(PDFContext[] pdfContexts)
01040           {
01041               HashSet<double> tbr = new HashSet<double>();
01042
01043               tbr.Add(1);
01044
```

```
01045             foreach (PDFContext ctx in pdfContexts)
01046             {
01047                 foreach (IFigure act in ctx._figures)
01048                 {
01049                     if (act.Stroke != null)
01050                     {
01051                         if (act.Stroke is SolidColourBrush solid)
01052                         {
01053                             tbr.Add(solid.A);
01054                         }
01055                         else if (act.Stroke is GradientBrush gradient)
01056                         {
01057                             foreach (GradientStop stop in gradient.GradientStops)
01058                             {
01059                                 tbr.Add(stop.Colour.A);
01060                             }
01061                         }
01062 
01063                     }
01064 
01065                     if (act.Fill != null)
01066                     {
01067                         if (act.Fill is SolidColourBrush solid)
01068                         {
01069                             tbr.Add(solid.A);
01070                         }
01071                         else if (act.Fill is GradientBrush gradient)
01072                         {
01073                             foreach (GradientStop stop in gradient.GradientStops)
01074                             {
01075                                 tbr.Add(stop.Colour.A);
01076                             }
01077                         }
01078                     }
01079                 }
01080             }
01081 
01082             return tbr.ToArray();
01083         }
01084 
01085         private static Dictionary<string, RasterImage> GetAllImages(PDFContext[] pdfContexts)
01086         {
01087             Dictionary<string, RasterImage> tbr = new Dictionary<string, RasterImage>();
01088 
01089             foreach (PDFContext ctx in pdfContexts)
01090             {
01091                 foreach (IFigure act in ctx._figures)
01092                 {
01093                     if (act is RasterImageFigure figure && !tbr.ContainsKey(figure.Image.Id))
01094                     {
01095                         tbr.Add(figure.Image.Id, figure.Image);
01096                     }
01097                 }
01098             }
01099 
01100             return tbr;
01101         }
01102 
01103 
01104         private static readonly char[] CP1252Chars = new char[] { '\u0000', '\u0001', '\u0002',
      '\u0003', '\u0004', '\u0005', '\u0006', '\u0007', '\u0008', '\u0009', '\u000A', '\u000B', '\u000C',
      '\u000D', '\u000E', '\u000F', '\u0010', '\u0011', '\u0012', '\u0013', '\u0014', '\u0015', '\u0016',
      '\u0017', '\u0018', '\u0019', '\u001A', '\u001B', '\u001C', '\u001D', '\u001E', '\u001F', '\u0020',
      '\u0021', '\u0022', '\u0023', '\u0024', '\u0025', '\u0026', '\u0027', '\u0028', '\u0029', '\u002A',
      '\u002B', '\u002C', '\u002D', '\u002E', '\u002F', '\u0030', '\u0031', '\u0032', '\u0033', '\u0034',
      '\u0035', '\u0036', '\u0037', '\u0038', '\u0039', '\u003A', '\u003B', '\u003C', '\u003D', '\u003E',
      '\u003F', '\u0040', '\u0041', '\u0042', '\u0043', '\u0044', '\u0045', '\u0046', '\u0047', '\u0048',
      '\u0049', '\u004A', '\u004B', '\u004C', '\u004D', '\u004E', '\u004F', '\u0050', '\u0051', '\u0052',
      '\u0053', '\u0054', '\u0055', '\u0056', '\u0057', '\u0058', '\u0059', '\u005A', '\u005B', '\u005C',
      '\u005D', '\u005E', '\u005F', '\u0060', '\u0061', '\u0062', '\u0063', '\u0064', '\u0065', '\u0066',
      '\u0067', '\u0068', '\u0069', '\u006A', '\u006B', '\u006C', '\u006D', '\u006E', '\u006F', '\u0070',
      '\u0071', '\u0072', '\u0073', '\u0074', '\u0075', '\u0076', '\u0077', '\u0078', '\u0079', '\u007A',
      '\u007B', '\u007C', '\u007D', '\u007E', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00A0', '\u00A1', '\u00A2',
      '\u00A3', '\u0000', '\u00A5', '\u0000', '\u00A7', '\u0000', '\u0000', '\u0000', '\u00AB', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00B5', '\u00B6',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u00BB', '\u0000', '\u0000', '\u00BE', '\u00BF', '\u00C0',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00C9', '\u0000',
      '\u0000', '\u00CC', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00D1', '\u00D2', '\u00D3', '\u00D4',
      '\u00D5', '\u00D6', '\u00D7', '\u00D8', '\u00D9', '\u00DA', '\u00DB', '\u00DC', '\u00DD', '\u00DE',
      '\u00DF', '\u00E0', '\u0000', '\u00E2', '\u0000', '\u00E4', '\u00E5', '\u00E6', '\u00E7', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u00EC', '\u00ED', '\u00EE', '\u00EF', '\u00F0', '\u0000', '\u00F2',
      '\u00F3', '\u00F4', '\u0000', '\u00F6', '\u00F7', '\u0000', '\u0000', '\u0000', '\u0000', '\u00FC',
      '\u00FD', '\u00FE', '\u00FF' };
01105 
```

```
01106 /// <summary>
01107 /// Save the document to a PDF file.
01108 /// </summary>
01109 /// <param name="document">The <see cref="Document"/> to save.</param>
01110 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
01111 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
01112 /// <param name="compressStreams">Indicates whether the streams in the PDF file should be
      compressed.</param>
01113 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
01114 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01115         public static void SaveAsPDF(this Document document, string fileName, TextOptions textOption =
      TextOptions.SubsetFonts, bool compressStreams = true, Dictionary<string, string> linkDestinations =
      null, FilterOption filterOption = default)
01116         {
01117             using (FileStream stream = new FileStream(fileName, FileMode.Create))
01118             {
01119                 document.SaveAsPDF(stream, textOption, compressStreams, linkDestinations,
      filterOption);
01120             }
01121         }
01122
01123 /// <summary>
01124 /// Defines whether the used fonts should be included in the file.
01125 /// </summary>
01126         public enum TextOptions
01127         {
01128 /// <summary>
01129 /// Embeds subsetted font files containing only the glyphs for the characters that have been used.
01130 /// </summary>
01131             SubsetFonts,
01132
01133 /// <summary>
01134 /// Does not embed any font file and converts all text items into paths.
01135 /// </summary>
01136             ConvertIntoPaths
01137         }
01138
01139 /// <summary>
01140 /// Determines how and whether image filters are rasterised.
01141 /// </summary>
01142         public class FilterOption
01143         {
01144 /// <summary>
01145 /// Defines whether image filters should be rasterised or not.
01146 /// </summary>
01147             public enum FilterOperations
01148             {
01149 /// <summary>
01150 /// Image filters will always be rasterised.
01151 /// </summary>
01152                 RasteriseAll,
01153
01154 /// <summary>
01155 /// All image filters will be ignored.
01156 /// </summary>
01157                 IgnoreAll,
01158
01159 /// <summary>
01160 /// All the images that should be drawn with a filter will be ignored.
01161 /// </summary>
01162                 SkipAll
01163             }
01164
01165 /// <summary>
01166 /// Defines whether image filters should be rasterised or not.
01167 /// </summary>
01168             public FilterOperations Operation { get; } = FilterOperations.RasteriseAll;
01169
01170 /// <summary>
01171 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
01172 /// </summary>
01173             public double RasterisationResolution { get; } = 1;
01174
01175 /// <summary>
01176 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
01177 /// </summary>
01178             public bool RasterisationResolutionRelative { get; } = true;
01179
```

```
01180 /// <summary>
01181 /// The default options for image filter rasterisation.
01182 /// </summary>
01183             public static FilterOption Default = new FilterOption(FilterOperations.RasteriseAll, 1,
      true);
01184 
01185 /// <summary>
01186 /// Create a new <see cref="FilterOption"/> object.
01187 /// </summary>
01188 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
01189 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
01190 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
01191             public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
01192             {
01193                 this.Operation = operation;
01194                 this.RasterisationResolution = rasterisationResolution;
01195                 this.RasterisationResolutionRelative = rasterisationResolutionRelative;
01196             }
01197         }
01198 
01199 
01200 
01201 /// <summary>
01202 /// Save the document to a PDF stream.
01203 /// </summary>
01204 /// <param name="document">The <see cref="Document"/> to save.</param>
01205 /// <param name="stream">The stream to which the PDF data will be written.</param>
01206 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
01207 /// <param name="compressStreams">Indicates whether the streams in the PDF file should be
      compressed.</param>
01208 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
01209 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01210         public static void SaveAsPDF(this Document document, Stream stream, TextOptions textOption =
      TextOptions.SubsetFonts, bool compressStreams = true, Dictionary<string, string> linkDestinations =
      null, FilterOption filterOption = default)
01211         {
01212             if (linkDestinations == null)
01213             {
01214                 linkDestinations = new Dictionary<string, string>();
01215             }
01216 
01217             if (filterOption == null)
01218             {
01219                 filterOption = FilterOption.Default;
01220             }
01221 
01222             long position = 0;
01223 
01224             List<long> objectPositions = new List<long>();
01225 
01226             int objectNum = 1;
01227             string currObject = "";
01228 
01229             int resourceObject = -1;
01230 
01231             StreamWriter sw = new StreamWriter(stream, Encoding.GetEncoding("ISO-8859-1"), 1024,
      true);
01232 
01233             //Header
01234             sw.Write("%PDF-1.4\n");
01235             position += 9;
01236 
01237             PDFContext[] pageContexts = new PDFContext[document.Pages.Count];
01238 
01239             for (int i = 0; i < document.Pages.Count; i++)
01240             {
01241                 pageContexts[i] = new PDFContext(document.Pages[i].Width, document.Pages[i].Height,
      document.Pages[i].Background, textOption == TextOptions.ConvertIntoPaths, filterOption);
01242                 document.Pages[i].Graphics.CopyToIGraphicsContext(pageContexts[i]);
01243             }
01244 
01245             Dictionary<string, FontFamily> allFontFamilies = GetFontFamilies(pageContexts);
01246             Dictionary<string, HashSet<char» usedChars = GetUsedChars(pageContexts);
01247             Dictionary<string, int> fontObjectNums = new Dictionary<string, int>();
01248             Dictionary<string, string> symbolFontIDs = new Dictionary<string, string>();
01249             Dictionary<string, string> nonSymbolFontIDs = new Dictionary<string, string>();
```

```
01250             Dictionary<string, Dictionary<char, int» symbolGlyphIndices = new Dictionary<string,
      Dictionary<char, int»();
01251             double[] alphas = GetAlphas(pageContexts);
01252             Dictionary<string, RasterImage> allImages = GetAllImages(pageContexts);
01253             Dictionary<string, int> imageObjectNums = new Dictionary<string, int>();
01254 
01255             int fontId = 1;
01256 
01257             foreach (KeyValuePair<string, FontFamily> kvp in allFontFamilies)
01258             {
01259                 List<char> nonSymbol = new List<char>();
01260                 List<char> symbol = new List<char>();
01261 
01262                 foreach (char c in usedChars[kvp.Key])
01263                 {
01264                     if (CP1252Chars.Contains(c))
01265                     {
01266                         nonSymbol.Add(c);
01267                     }
01268                     else
01269                     {
01270                         symbol.Add(c);
01271                     }
01272                 }
01273 
01274                 //Font
01275                 if (((kvp.Value.IsStandardFamily && kvp.Value.FileName != "Symbol" &&
      kvp.Value.FileName != "ZapfDingbats") && symbol.Count == 0) || kvp.Value.TrueTypeFile == null)
01276                 {
01277                     fontObjectNums.Add("nonsymbol:  " + kvp.Key, objectNum);
01278                     fontObjectNums.Add("symbol:  " + kvp.Key, objectNum);
01279                     nonSymbolFontIDs.Add(kvp.Key, "F" + fontId.ToString());
01280                     symbolFontIDs.Add(kvp.Key, "F" + fontId.ToString());
01281                     objectPositions.Add(position);
01282                     currObject = objectNum.ToString() + " 0 obj\n« /Type /Font /Subtype /Type1
      /BaseFont /" + (kvp.Value.IsStandardFamily ?  kvp.Value.FileName.Replace(" ", "-") :  kvp.Key) + "
      »\nendobj\n";
01283                     sw.Write(currObject);
01284                     position += currObject.Length;
01285                     objectNum++;
01286                     fontId++;
01287                 }
01288                 else
01289                 {
01290                     int fontFileInd = objectNum;
01291 
01292                     TrueTypeFile subsettedFont = kvp.Value.TrueTypeFile.SubsetFont(new
      string(usedChars[kvp.Key].ToArray()));
01293 
01294                     Stream compressedStream;
01295 
01296                     if (!compressStreams)
01297                     {
01298                         compressedStream = subsettedFont.FontStream;
01299                     }
01300                     else
01301                     {
01302                         compressedStream = ZLibCompress(subsettedFont.FontStream);
01303                     }
01304 
01305                     long length = compressedStream.Length;
01306 
01307                     objectPositions.Add(position);
01308                     currObject = objectNum.ToString() + " 0 obj\n« /Length " + length.ToString() + "
      /Length1 " + subsettedFont.FontStream.Length.ToString();
01309 
01310                     if (compressStreams)
01311                     {
01312                         currObject += " /Filter [ /FlateDecode ]";
01313                     }
01314 
01315                     currObject += " »\nstream\n";
01316                     sw.Write(currObject);
01317                     position += currObject.Length;
01318                     sw.Flush();
01319 
01320                     compressedStream.Seek(0, SeekOrigin.Begin);
01321                     compressedStream.CopyTo(stream);
01322 
01323                     position += length;
01324                     currObject = "endstream\nendobj\n";
01325                     sw.Write(currObject);
01326                     position += currObject.Length;
01327                     objectNum++;
01328 
01329 
01330                     if (nonSymbol.Count > 0)
```

```
01331                   {
01332                       PDFFontDescriptor desc = new PDFFontDescriptor(subsettedFont, true, false);
01333
01334                       int fontDescriptorInd = objectNum;
01335                       objectPositions.Add(position);
01336
01337                       currObject = objectNum.ToString() + " 0 obj\n« /Type /FontDescriptor /FontName
      /" + desc.FontName + " /FontFamily (" + EscapeStringForPDF(desc.FontFamily) + ") /Flags " +
      desc.Flags.ToString();
01338                       currObject += " /FontBBox [ " +
      desc.FontBBox[0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      " + desc.FontBBox[1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture)
      + " " + desc.FontBBox[2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      desc.FontBBox[3].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      ] /ItalicAngle " + desc.ItalicAngle.ToString();
01339                       currObject += " /Ascent " + desc.Ascent.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " /Descent " +
      desc.Descent.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      /CapHeight " + desc.CapHeight.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " /StemV " + desc.StemV.ToString() + " /StemH " +
      desc.StemH.ToString() + " /FontFile2 " + fontFileInd.ToString() + " 0 R »\nendobj\n";
01340                       sw.Write(currObject);
01341                       position += currObject.Length;
01342                       objectNum++;
01343
01344
01345                       fontObjectNums.Add("nonsymbol:  " + kvp.Key, objectNum);
01346                       nonSymbolFontIDs.Add(kvp.Key, "F" + fontId.ToString());
01347                       objectPositions.Add(position);
01348
01349                       int firstChar = (from el in nonSymbol select Array.IndexOf(CP1252Chars,
      el)).Min();
01350                       int lastChar = (from el in nonSymbol select Array.IndexOf(CP1252Chars,
      el)).Max();
01351
01352                       currObject = objectNum.ToString() + " 0 obj\n« /Type /Font /Subtype /TrueType
      /BaseFont /" + desc.FontName + " /FirstChar " + firstChar.ToString() + " /LastChar " +
      lastChar.ToString() + " /FontDescriptor " + fontDescriptorInd.ToString() + " 0 R /Encoding
      /WinAnsiEncoding /Widths [ ";
01353
01354                       for (int i = firstChar; i <= lastChar; i++)
01355                       {
01356                           if (nonSymbol.Contains(CP1252Chars[i]))
01357                           {
01358                               currObject +=
      subsettedFont.Get1000EmGlyphWidth(CP1252Chars[i]).ToString() + " ";
01359                           }
01360                           else
01361                           {
01362                               currObject += "0 ";
01363                           }
01364                       }
01365
01366                       currObject += "] »\nendobj\n";
01367                       sw.Write(currObject);
01368                       position += currObject.Length;
01369                       objectNum++;
01370                       fontId++;
01371                   }
01372
01373
01374                   if (symbol.Count > 0)
01375                   {
01376                       PDFFontDescriptor desc = new PDFFontDescriptor(subsettedFont, true, true);
01377
01378                       int fontDescriptorInd = objectNum;
01379                       objectPositions.Add(position);
01380
01381                       currObject = objectNum.ToString() + " 0 obj\n« /Type /FontDescriptor /FontName
      /" + desc.FontName + " /FontFamily (" + EscapeStringForPDF(desc.FontFamily) + ") /Flags " +
      desc.Flags.ToString();
01382                       currObject += " /FontBBox [ " +
      desc.FontBBox[0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      " + desc.FontBBox[1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture)
      + " " + desc.FontBBox[2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      desc.FontBBox[3].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      ] /ItalicAngle " + desc.ItalicAngle.ToString();
01383                       currObject += " /Ascent " + desc.Ascent.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " /Descent " +
      desc.Descent.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      /CapHeight " + desc.CapHeight.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " /StemV " + desc.StemV.ToString() + " /StemH " +
      desc.StemH.ToString() + " /FontFile2 " + fontFileInd.ToString() + " 0 R »\nendobj\n";
01384                       sw.Write(currObject);
01385                       position += currObject.Length;
```

```
01386                           objectNum++;
01387
01388
01389                           Dictionary<char, int> glyphIndices = new Dictionary<char, int>();
01390
01391                           for (int i = 0; i < symbol.Count; i++)
01392                           {
01393                               glyphIndices.Add(symbol[i], subsettedFont.GetGlyphIndex(symbol[i]));
01394                           }
01395
01396                           symbolGlyphIndices.Add(kvp.Key, glyphIndices);
01397
01398                           int descendantFontInd = objectNum;
01399                           objectPositions.Add(position);
01400                           currObject = objectNum.ToString() + " 0 obj\n« /Type /Font /Subtype
      /CIDFontType2 /BaseFont /" + desc.FontName + " /CIDSystemInfo « /Registry (Adobe) /Ordering (Identity)
      /Supplement 0 » /FontDescriptor " + fontDescriptorInd.ToString() + " 0 R ";
01401                           currObject += "/W [ ";
01402
01403                           for (int i = 0; i < symbol.Count; i++)
01404                           {
01405                               currObject += glyphIndices[symbol[i]].ToString() + " [ ";
01406                               currObject += subsettedFont.Get1000EmGlyphWidth(symbol[i]).ToString() + "
      ] ";
01407                           }
01408
01409                           currObject += "] »\nendobj\n";
01410                           sw.Write(currObject);
01411                           position += currObject.Length;
01412                           objectNum++;
01413
01414
01415                           string toUnicodeStream = "/CIDInit /ProcSet findresource begin\n12 dict
      begin\nbegincmap\n/CIDSystemInfo « /Registry (Adobe) /Ordering (UCS) /Supplement 0 » def\n";
01416                           toUnicodeStream += "/CMapName /Adobe-Identity-UCS def\n/CMapType 2 def\n1
      begincodespacerange\n<0000> <ffff>\nendcodespacerange\n1 beginbfchar\n";
01417                           for (int i = 0; i < symbol.Count; i++)
01418                           {
01419                               toUnicodeStream += "<" + glyphIndices[symbol[i]].ToString("X4") + "> <" +
      ((int)symbol[i]).ToString("X4") + ">\n";
01420                           }
01421                           toUnicodeStream += "endbfchar\nendcmap\nCmapName currentdict /CMap
      defineresource pop\nend\nend\n";
01422
01423                           MemoryStream uncompressedUnicode = new MemoryStream();
01424
01425                           using (StreamWriter usw = new StreamWriter(uncompressedUnicode,
      Encoding.ASCII, 1024, true))
01426                           {
01427                               usw.Write(toUnicodeStream);
01428                           }
01429
01430                           uncompressedUnicode.Seek(0, SeekOrigin.Begin);
01431
01432                           MemoryStream compressedToUnicode;
01433
01434                           if (!compressStreams)
01435                           {
01436                               compressedToUnicode = uncompressedUnicode;
01437                           }
01438                           else
01439                           {
01440                               compressedToUnicode = ZLibCompress(uncompressedUnicode);
01441                           }
01442
01443                           long unicodeLength = compressedToUnicode.Length;
01444
01445                           int toUnicodeInd = objectNum;
01446                           objectPositions.Add(position);
01447                           currObject = objectNum.ToString() + " 0 obj\n« /Length " + unicodeLength;
01448
01449                           if (compressStreams)
01450                           {
01451                               currObject += " /Filter [ /FlateDecode ]";
01452                           }
01453
01454                           currObject += " »\nstream\n";
01455
01456                           sw.Write(currObject);
01457                           position += currObject.Length;
01458                           sw.Flush();
01459
01460                           compressedToUnicode.WriteTo(stream);
01461                           position += unicodeLength;
01462
01463                           currObject = "endstream\nendobj\n";
01464                           sw.Write(currObject);
```

```
01465                         position += currObject.Length;
01466                         objectNum++;
01467 
01468 
01469                         fontObjectNums.Add("symbol:  " + kvp.Key, objectNum);
01470                         symbolFontIDs.Add(kvp.Key, "F" + fontId.ToString());
01471                         objectPositions.Add(position);
01472                         currObject = objectNum.ToString() + " 0 obj\n« /Type /Font /Subtype /Type0
      /BaseFont /" + desc.FontName + " /Encoding /Identity-H /DescendantFonts [ " +
      descendantFontInd.ToString() + " 0 R ] /ToUnicode " + toUnicodeInd.ToString() + " 0 R »\nendobj\n";
01473                         sw.Write(currObject);
01474                         position += currObject.Length;
01475                         objectNum++;
01476                         fontId++;
01477                     }
01478                 }
01479             }
01480 
01481             foreach (KeyValuePair<string, RasterImage> img in allImages)
01482             {
01483                 RasterImage image = img.Value;
01484                 int stride = image.Width * (image.HasAlpha ?  4 :  3);
01485 
01486                 string filter = "";
01487 
01488                 if (image.HasAlpha)
01489                 {
01490                     objectPositions.Add(position);
01491 
01492                     filter = "";
01493                     MemoryStream alphaStream = new MemoryStream();
01494 
01495                     unsafe
01496                     {
01497                         byte* dataPointer = (byte*)image.ImageDataAddress;
01498 
01499                         if (compressStreams)
01500                         {
01501                             filter = "/FlateDecode";
01502 
01503                             for (int y = 0; y < image.Height; y++)
01504                             {
01505                                 for (int x = 0; x < image.Width; x++)
01506                                 {
01507                                     dataPointer += 3;
01508                                     alphaStream.WriteByte(*dataPointer);
01509                                     dataPointer++;
01510                                 }
01511                             }
01512 
01513                             alphaStream.Seek(0, SeekOrigin.Begin);
01514                             MemoryStream compressed = ZLibCompress(alphaStream);
01515                             alphaStream.Dispose();
01516                             alphaStream = compressed;
01517                         }
01518                         else
01519                         {
01520                             filter = "/ASCIIHexDecode";
01521 
01522                             using (StreamWriter imageWriter = new StreamWriter(alphaStream,
      Encoding.ASCII, 1024, true))
01523                             {
01524                                 for (int y = 0; y < image.Height; y++)
01525                                 {
01526                                     for (int x = 0; x < image.Width; x++)
01527                                     {
01528                                         dataPointer += 3;
01529                                         imageWriter.Write((*dataPointer).ToString("X2"));
01530                                         dataPointer++;
01531                                     }
01532                                 }
01533                             }
01534                         }
01535                     }
01536 
01537                     currObject = objectNum.ToString() + " 0 obj\n« /Type /XObject /Subtype /Image
      /Width " + image.Width.ToString() + " /Height " + image.Height.ToString() + " /ColorSpace /DeviceGray
      /BitsPerComponent 8 /Interpolate " + (image.Interpolate ?  "true" :  "false") + " /Filter " + filter +
      " /Length " + alphaStream.Length + " »\nstream\n";
01538 
01539                     sw.Write(currObject);
01540                     position += currObject.Length;
01541                     sw.Flush();
01542 
01543                     alphaStream.Seek(0, SeekOrigin.Begin);
01544                     alphaStream.CopyTo(stream);
01545                     position += alphaStream.Length;
```

```
01546
01547                       currObject = "\nendstream\nendobj\n";
01548                       sw.Write(currObject);
01549                       position += currObject.Length;
01550                       objectNum++;
01551
01552                       alphaStream.Dispose();
01553
01554                   }
01555
01556                   objectPositions.Add(position);
01557                   int imageObjectNum = objectNum;
01558
01559                   filter = "";
01560                   MemoryStream imageStream = new MemoryStream();
01561
01562                   unsafe
01563                   {
01564                       byte* dataPointer = (byte*)image.ImageDataAddress;
01565
01566                       if (compressStreams)
01567                       {
01568                           filter = "/FlateDecode";
01569
01570                           if (image.HasAlpha)
01571                           {
01572                               for (int y = 0; y < image.Height; y++)
01573                               {
01574                                   for (int x = 0; x < image.Width; x++)
01575                                   {
01576                                       imageStream.WriteByte(*dataPointer);
01577                                       dataPointer++;
01578                                       imageStream.WriteByte(*dataPointer);
01579                                       dataPointer++;
01580                                       imageStream.WriteByte(*dataPointer);
01581                                       dataPointer++;
01582                                       dataPointer++;
01583                                   }
01584                               }
01585                           }
01586                           else
01587                           {
01588                               for (int y = 0; y < image.Height; y++)
01589                               {
01590                                   for (int x = 0; x < image.Width; x++)
01591                                   {
01592                                       imageStream.WriteByte(*dataPointer);
01593                                       dataPointer++;
01594                                       imageStream.WriteByte(*dataPointer);
01595                                       dataPointer++;
01596                                       imageStream.WriteByte(*dataPointer);
01597                                       dataPointer++;
01598                                   }
01599                               }
01600                           }
01601
01602                           imageStream.Seek(0, SeekOrigin.Begin);
01603                           MemoryStream compressed = ZLibCompress(imageStream);
01604                           imageStream.Dispose();
01605                           imageStream = compressed;
01606                       }
01607                       else
01608                       {
01609                           filter = "/ASCIIHexDecode";
01610
01611                           using (StreamWriter imageWriter = new StreamWriter(imageStream,
      Encoding.ASCII, 1024, true))
01612                           {
01613                               if (image.HasAlpha)
01614                               {
01615                                   for (int y = 0; y < image.Height; y++)
01616                                   {
01617                                       for (int x = 0; x < image.Width; x++)
01618                                       {
01619                                           imageWriter.Write((*dataPointer).ToString("X2"));
01620                                           dataPointer++;
01621                                           imageWriter.Write((*dataPointer).ToString("X2"));
01622                                           dataPointer++;
01623                                           imageWriter.Write((*dataPointer).ToString("X2"));
01624                                           dataPointer++;
01625                                           dataPointer++;
01626                                       }
01627                                   }
01628                               }
01629                               else
01630                               {
01631                                   for (int y = 0; y < image.Height; y++)
```

```
01632                                   {
01633                                       for (int x = 0; x < image.Width; x++)
01634                                       {
01635                                           imageWriter.Write((*dataPointer).ToString("X2"));
01636                                           dataPointer++;
01637                                           imageWriter.Write((*dataPointer).ToString("X2"));
01638                                           dataPointer++;
01639                                           imageWriter.Write((*dataPointer).ToString("X2"));
01640                                           dataPointer++;
01641                                       }
01642                                   }
01643                               }
01644                           }
01645                       }
01646                   }
01647
01648                   currObject = objectNum.ToString() + " 0 obj\n« /Type /XObject /Subtype /Image /Width "
      + image.Width.ToString() + " /Height " + image.Height.ToString() + " /ColorSpace /DeviceRGB
      /BitsPerComponent 8 /Interpolate " + (image.Interpolate ?  "true" :  "false") + " /Filter " + filter +
      " /Length " + imageStream.Length + (image.HasAlpha ?  " /SMask " + (objectNum - 1) + " 0 R" :  "") + "
      »\nstream\n";
01649
01650                   sw.Write(currObject);
01651                   position += currObject.Length;
01652                   sw.Flush();
01653
01654                   imageStream.Seek(0, SeekOrigin.Begin);
01655                   imageStream.CopyTo(stream);
01656                   position += imageStream.Length;
01657
01658                   currObject = "\nendstream\nendobj\n";
01659                   sw.Write(currObject);
01660                   position += currObject.Length;
01661                   objectNum++;
01662
01663                   imageStream.Dispose();
01664
01665                   imageObjectNums.Add(img.Key, imageObjectNum);
01666               }
01667
01668               int fontListObject = -1;
01669
01670               if (allFontFamilies.Count > 0)
01671               {
01672                   //Fonts
01673                   objectPositions.Add(position);
01674                   fontListObject = objectNum;
01675                   currObject = objectNum.ToString() + " 0 obj\n« ";
01676                   foreach (KeyValuePair<string, string> kvp in nonSymbolFontIDs)
01677                   {
01678                       currObject += "/" + kvp.Value + " " + fontObjectNums["nonsymbol:  " +
      kvp.Key].ToString() + " 0 R ";
01679                   }
01680                   foreach (KeyValuePair<string, string> kvp in symbolFontIDs)
01681                   {
01682                       currObject += "/" + kvp.Value + " " + fontObjectNums["symbol:  " +
      kvp.Key].ToString() + " 0 R ";
01683                   }
01684                   currObject += "»\nendobj\n";
01685                   sw.Write(currObject);
01686                   position += currObject.Length;
01687                   objectNum++;
01688               }
01689
01690
01691               int[] pageContentInd = new int[document.Pages.Count];
01692
01693
01694               List<(string, List<(double, double, double, double)>)>[] taggedObjectRectsByPage = new
      List<(string, List<(double, double, double, double)>)>[document.Pages.Count];
01695               Dictionary<string, int>[] taggedObjectRectsIndicesByPage = new Dictionary<string,
      int>[document.Pages.Count];
01696               List<(GradientBrush, double[,], IFigure)> gradients = new
      System.Collections.Generic.List<(GradientBrush, double[,], IFigure)>();
01697
01698
01699               for (int pageInd = 0; pageInd < document.Pages.Count; pageInd++)
01700               {
01701                   taggedObjectRectsByPage[pageInd] = new List<(string, List<(double, double, double,
      double)>)>();
01702                   taggedObjectRectsIndicesByPage[pageInd] = new Dictionary<string, int>();
01703
01704                   double[,] transformationMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0,
      1 } };
01705                   Stack<double[,]> savedStates = new Stack<double[,]>();
01706
01707                   MemoryStream contentStream = new MemoryStream();
```

```
01708
01709                 using (StreamWriter ctW = new StreamWriter(contentStream, Encoding.ASCII, 1024, true))
01710                 {
01711                     for (int i = 0; i < pageContexts[pageInd]._figures.Count; i++)
01712                     {
01713                         bool isTransform = pageContexts[pageInd]._figures[i] is TransformFigure;
01714
01715                         bool isClip = pageContexts[pageInd]._figures[i] is PathFigure pathFig &&
      pathFig.IsClipping;
01716
01717                         if (!string.IsNullOrEmpty(pageContexts[pageInd]._figures[i].Tag) &&
      !isTransform && !isClip)
01718                         {
01719                             (double, double, double, double) boundingRect =
      MeasureFigure(pageContexts[pageInd]._figures[i], ref transformationMatrix, savedStates);
01720
01721                             if
      (!taggedObjectRectsIndicesByPage[pageInd].TryGetValue(pageContexts[pageInd]._figures[i].Tag, out int
      index))
01722                             {
01723
      taggedObjectRectsByPage[pageInd].Add((pageContexts[pageInd]._figures[i].Tag, new List<(double, double,
      double, double)> { boundingRect }));
01724
      taggedObjectRectsIndicesByPage[pageInd][pageContexts[pageInd]._figures[i].Tag] =
      taggedObjectRectsByPage[pageInd].Count - 1;
01725                             }
01726                             else
01727                             {
01728                                 (string, List<(double, double, double, double)>) previousRect =
      taggedObjectRectsByPage[pageInd][index];
01729                                 taggedObjectRectsByPage[pageInd][index].Item2.Add(boundingRect);
01730                             }
01731                         }
01732                         else if (isTransform)
01733                         {
01734                             MeasureFigure(pageContexts[pageInd]._figures[i], ref transformationMatrix,
      savedStates);
01735                         }
01736
01737                         ctW.Write(FigureAsPDFString(pageContexts[pageInd]._figures[i],
      nonSymbolFontIDs, symbolFontIDs, symbolGlyphIndices, alphas, imageObjectNums, transformationMatrix,
      gradients));
01738                     }
01739                 }
01740
01741                 //Contents
01742                 objectPositions.Add(position);
01743                 contentStream.Seek(0, SeekOrigin.Begin);
01744
01745                 MemoryStream compressedStream;
01746
01747                 if (!compressStreams)
01748                 {
01749                     compressedStream = contentStream;
01750                 }
01751                 else
01752                 {
01753                     compressedStream = ZLibCompress(contentStream);
01754                 }
01755
01756                 long streamLength = compressedStream.Length;
01757
01758                 pageContentInd[pageInd] = objectNum;
01759                 currObject = objectNum.ToString() + " 0 obj\n« /Length " +
      streamLength.ToString(System.Globalization.CultureInfo.InvariantCulture);
01760
01761                 if (compressStreams)
01762                 {
01763                     currObject += " /Filter [ /FlateDecode ]";
01764                 }
01765
01766                 currObject += " »\nstream\n";
01767
01768                 sw.Write(currObject);
01769                 sw.Flush();
01770
01771                 position += currObject.Length;
01772                 compressedStream.WriteTo(stream);
01773                 position += streamLength;
01774
01775                 compressedStream.Dispose();
01776
01777                 currObject = "endstream\nendobj\n";
01778                 sw.Write(currObject);
01779                 position += currObject.Length;
01780
```

```
01781                 objectNum++;
01782             }
01783 
01784             List<int> gradientIndices = new List<int>(gradients.Count);
01785             List<int> gradientAlphaIndices = new List<int>(gradients.Count);
01786             List<int> gradientMaskIndices = new List<int>(gradients.Count);
01787 
01788             if (gradients.Count > 0)
01789             {
01790                 for (int i = 0; i < gradients.Count; i++)
01791                 {
01792                     (GradientBrush gradient, double[,] matrix, IFigure figure) = gradients[i];
01793 
01794                     //int functionObject = -1;
01795 
01796                     bool hasAlpha = false;
01797 
01798                     /*if (gradient.GradientStops.Count == 2)
01799 {
01800 objectPositions.Add(position);
01801 
01802 currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 2 /Domain [ 0 1 ] /C0 [ " +
      gradient.GradientStops[0].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[0].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[0].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ]
      ";
01803 currObject += "/C1 [ " +
      gradient.GradientStops[1].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[1].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[1].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ]
      /N 1 »\nendobj\n";
01804 sw.Write(currObject);
01805 functionObject = objectNum;
01806 
01807 position += currObject.Length;
01808 objectNum++;
01809 
01810 hasAlpha = gradient.GradientStops[0].Colour.A != 1 || gradient.GradientStops[1].Colour.A != 1;
01811 }
01812 else
01813 {
01814 List<double> bounds = new List<double>();
01815 List<int> functionIndices = new List<int>();
01816 
01817 for (int j = 0; j < gradient.GradientStops.Count - 1; j++)
01818 {
01819 objectPositions.Add(position);
01820 
01821 currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 2 /Domain [ 0 1 ] /C0 [ " +
      gradient.GradientStops[j].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[j].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[j].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ]
      ";
01822 currObject += "/C1 [ " + gradient.GradientStops[j +
      1].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[j + 1].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + "
      " + gradient.GradientStops[j + 1].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " ] /N 1 »\nendobj\n";
01823 sw.Write(currObject);
01824 functionIndices.Add(objectNum);
01825 
01826 if (j < gradient.GradientStops.Count - 2)
01827 {
01828 bounds.Add(gradient.GradientStops[j + 1].Offset);
01829 }
01830 
01831 position += currObject.Length;
01832 objectNum++;
01833 
01834 if (gradient.GradientStops[j].Colour.A != 1)
01835 {
01836 hasAlpha = true;
01837 }
01838 }
01839 
01840 objectPositions.Add(position);
01841 
01842 currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 3 /Domain [ 0 1 ] /Functions [ ";
01843 
01844 for (int j = 0; j < functionIndices.Count; j++)
01845 {
01846 currObject += functionIndices[j].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R
      ";
01847 }
01848 
01849 currObject += "] /Bounds [ ";
01850 
```

```
01851 for (int j = 0; j < bounds.Count; j++)
01852 {
01853 currObject += bounds[j].ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
01854 }
01855
01856 currObject += "] /Encode [ ";
01857
01858 for (int j = 0; j < functionIndices.Count; j++)
01859 {
01860 currObject += "0 1 ";
01861 }
01862
01863 currObject += "] »\nendobj\n";
01864
01865
01866 sw.Write(currObject);
01867 functionObject = objectNum;
01868
01869 position += currObject.Length;
01870 objectNum++;
01871
01872 }
01873
01874 if (gradient is LinearGradientBrush linear)
01875 {
01876 objectPositions.Add(position);
01877
01878 currObject = objectNum.ToString() + " 0 obj\n« /Type /Pattern /PatternType 2 /Matrix [ " +
01879
01880 matrix[0, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01881 matrix[1, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01882 matrix[0, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01883 matrix[1, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01884 matrix[0, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01885 matrix[1, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " ]
      ";
01886
01887 currObject += "/Shading « /ShadingType 2 /ColorSpace /DeviceRGB /Coords [ " +
      linear.StartPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.StartPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.EndPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.EndPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ] ";
01888
01889 currObject += "/Domain [ 0 1 ] /Extend [ true true ] /Function " +
      functionObject.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R » »\nendobj\n";
01890 sw.Write(currObject);
01891 gradientIndices.Add(objectNum);
01892
01893 position += currObject.Length;
01894 objectNum++;
01895 }
01896 else if (gradient is RadialGradientBrush radial)
01897 {
01898 objectPositions.Add(position);
01899
01900 currObject = objectNum.ToString() + " 0 obj\n« /Type /Pattern /PatternType 2 /Matrix [ " +
01901
01902 matrix[0, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01903 matrix[1, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01904 matrix[0, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01905 matrix[1, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01906 matrix[0, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01907 matrix[1, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " ]
      ";
01908
01909 currObject += "/Shading « /ShadingType 3 /ColorSpace /DeviceRGB /Coords [ " +
      radial.FocalPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.FocalPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 " +
      radial.Centre.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.Centre.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.Radius.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ] ";
01910
01911 currObject += "/Domain [ 0 1 ] /Extend [ true true ] /Function " +
      functionObject.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R » »\nendobj\n";
01912 sw.Write(currObject);
01913 gradientIndices.Add(objectNum);
01914
01915 position += currObject.Length;
01916 objectNum++;
01917 }*/
01918
01919                     WriteGradient(true, ref gradient, ref objectPositions, ref position, ref
      currObject, ref objectNum, ref sw, ref hasAlpha, ref matrix, ref gradientIndices);
01920
01921                     if (!hasAlpha)
01922                     {
01923                         gradientAlphaIndices.Add(-1);
```

```
01924                           gradientMaskIndices.Add(-1);
01925                       }
01926                       else
01927                       {
01928                           /*objectPositions.Add(position);
01929 int alphaGradientIndex = objectNum;
01930
01931
01932 **/
01933
01934                           GradientBrush alphaGradient =
      (GradientBrush)PDFContext.GetAlphaBrush(gradient);
01935
01936                           bool hasAlpha2 = false;
01937
01938                           WriteGradient(false, ref alphaGradient, ref objectPositions, ref position, ref
      currObject, ref objectNum, ref sw, ref hasAlpha2, ref matrix, ref gradientAlphaIndices);
01939
01940                           int alphaGradientIndex = gradientAlphaIndices[gradientAlphaIndices.Count - 1];
01941
01942                           Rectangle bbox = figure.GetBounds();
01943
01944                           MemoryStream contentStream = new MemoryStream();
01945
01946                           using (StreamWriter ctW = new StreamWriter(contentStream, Encoding.ASCII,
      1024, true))
01947                           {
01948                               ctW.Write("q\n");
01949                               ctW.Write(bbox.Location.X.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      bbox.Location.Y.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      " + bbox.Size.Width.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture)
      + " " + bbox.Size.Height.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " re\n");
01950                               ctW.Write("/Pattern cs\n");
01951                               ctW.Write("/pa" + gradientAlphaIndices.Count + " scn\n");
01952                               ctW.Write("f\n");
01953                               ctW.Write("Q\n");
01954                           }
01955
01956                           contentStream.Seek(0, SeekOrigin.Begin);
01957
01958                           MemoryStream compressedStream;
01959
01960                           if (!compressStreams)
01961                           {
01962                               compressedStream = contentStream;
01963                           }
01964                           else
01965                           {
01966                               compressedStream = ZLibCompress(contentStream);
01967                           }
01968
01969                           long streamLength = compressedStream.Length;
01970
01971                           objectPositions.Add(position);
01972                           int maskIndex = objectNum;
01973
01974                           currObject = objectNum.ToString() + " 0 obj\n« /Type /XObject /Subtype /Form "
      +
01975                               "/Group « /Type /Group /S /Transparency /I true /CS /DeviceRGB » " +
01976                               "/BBox [ " +
01977                               bbox.Location.X.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      bbox.Location.Y.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      " + (bbox.Location.X + bbox.Size.Width).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + (bbox.Location.Y +
      bbox.Size.Height).ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) +
01978                               " ] " + /*"/Matrix [ " +
01979 matrix[0, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01980 matrix[1, 0].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01981 matrix[0, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01982 matrix[1, 1].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01983 matrix[0, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " " +
01984 matrix[1, 2].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " ] "
      +*/
01985                           "/Resources « /Pattern « /pa" + gradientAlphaIndices.Count + " " +
      alphaGradientIndex.ToString() + " 0 R » » " +
01986
01987                           "/Length " + streamLength.ToString();
01988
01989                           if (compressStreams)
01990                           {
01991                               if (compressStreams)
01992                               {
01993                                   currObject += " /Filter [ /FlateDecode ]";
01994                               }
```

```
01995                         }
01996
01997                         currObject += " »\nstream\n";
01998
01999                         sw.Write(currObject);
02000                         sw.Flush();
02001
02002                         position += currObject.Length;
02003                         compressedStream.WriteTo(stream);
02004                         position += streamLength;
02005
02006                         compressedStream.Dispose();
02007
02008                         currObject = "endstream\nendobj\n";
02009                         sw.Write(currObject);
02010                         position += currObject.Length;
02011
02012                         objectNum++;
02013
02014
02015                         objectPositions.Add(position);
02016                         int actualMaskIndex = objectNum;
02017
02018                         gradientMaskIndices.Add(actualMaskIndex);
02019
02020                         currObject = objectNum.ToString() + " 0 obj\n« /Type /ExtGState /SMask « /Type
      /Mask /S /Luminosity /G " + maskIndex.ToString() + " 0 R » »\nendobj\n";
02021                         sw.Write(currObject);
02022                         position += currObject.Length;
02023                         objectNum++;
02024                     }
02025
02026                 }
02027             }
02028
02029             if (allFontFamilies.Count > 0)
02030             {
02031
02032                 //Resources
02033                 objectPositions.Add(position);
02034                 resourceObject = objectNum;
02035                 currObject = objectNum.ToString() + " 0 obj\n« /Font " + fontListObject.ToString() + "
      0 R";
02036
02037                 if (alphas.Length > 0 || gradientMaskIndices.Where(x => x >= 0).Any())
02038                 {
02039                     currObject += " /ExtGState «\n";
02040
02041                     for (int i = 0; i < alphas.Length; i++)
02042                     {
02043                         currObject += "/a" + i.ToString() + " « /CA " +
      alphas[i].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " /ca "
      + alphas[i].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      »\n";
02044                     }
02045
02046                     for (int i = 0; i < gradientMaskIndices.Count; i++)
02047                     {
02048                         if (gradientMaskIndices[i] >= 0)
02049                         {
02050                             currObject += "/ma" + i.ToString() + " " +
      gradientMaskIndices[i].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R\n";
02051                         }
02052                     }
02053
02054                     currObject += "»";
02055                 }
02056
02057                 if (imageObjectNums.Count > 0)
02058                 {
02059                     currObject += " /XObject «";
02060
02061                     foreach (KeyValuePair<string, int> kvp in imageObjectNums)
02062                     {
02063                         currObject += " /Img" + kvp.Value.ToString() + " " + kvp.Value.ToString() + "
      0 R";
02064                     }
02065
02066                     currObject += " »";
02067                 }
02068
02069                 if (gradientIndices.Count > 0)
02070                 {
02071                     currObject += " /Pattern « ";
02072
02073                     for (int i = 0; i < gradientIndices.Count; i++)
02074                     {
```

```
02075                       currObject += "/p" +
      i.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradientIndices[i].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R ";
02076                   }
02077
02078                   currObject += "»";
02079               }
02080
02081               currObject += " »\nendobj\n";
02082               sw.Write(currObject);
02083               position += currObject.Length;
02084               objectNum++;
02085           }
02086           else
02087           {
02088               //Resources
02089               objectPositions.Add(position);
02090               resourceObject = objectNum;
02091               currObject = objectNum.ToString() + " 0 obj\n«";
02092
02093               if (alphas.Length > 0 || gradientMaskIndices.Where(x => x >= 0).Any())
02094               {
02095                   currObject += " /ExtGState «\n";
02096
02097                   for (int i = 0; i < alphas.Length; i++)
02098                   {
02099                       currObject += "/a" + i.ToString() + " « /CA " +
      alphas[i].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + " /ca "
      + alphas[i].ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) + "
      »\n";
02100                   }
02101
02102                   for (int i = 0; i < gradientMaskIndices.Count; i++)
02103                   {
02104                       if (gradientMaskIndices[i] >= 0)
02105                       {
02106                           currObject += "/ma" + i.ToString() + " " +
      gradientMaskIndices[i].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R\n";
02107                       }
02108                   }
02109
02110                   currObject += "»";
02111               }
02112
02113               if (imageObjectNums.Count > 0)
02114               {
02115                   currObject += " /XObject «";
02116
02117                   foreach (KeyValuePair<string, int> kvp in imageObjectNums)
02118                   {
02119                       currObject += " /Img" + kvp.Value.ToString() + " " + kvp.Value.ToString() + "
      0 R";
02120                   }
02121
02122                   currObject += " »";
02123               }
02124
02125               if (gradientIndices.Count > 0)
02126               {
02127                   currObject += " /Pattern « ";
02128
02129                   for (int i = 0; i < gradientIndices.Count; i++)
02130                   {
02131                       currObject += "/p" +
      i.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradientIndices[i].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R ";
02132                   }
02133
02134                   currObject += "»";
02135               }
02136
02137               currObject += " »\nendobj\n";
02138               sw.Write(currObject);
02139               position += currObject.Length;
02140               objectNum++;
02141           }
02142
02143           //Catalog
02144           objectPositions.Add(position);
02145           int rootObject = objectNum;
02146           currObject = objectNum.ToString() + " 0 obj\n« /Type /Catalog /Pages " + (objectNum +
      1).ToString() + " 0 R »\nendobj\n";
02147           sw.Write(currObject);
02148           position += currObject.Length;
02149           objectNum++;
02150
02151           objectPositions.Add(position);
```

```
02152             int pageParent = objectNum;
02153             objectNum++;
02154 
02155             List<int> pageObjectNums = new List<int>();
02156 
02157             //We do not have enough information to resolve all relative links yet (we need the object
      number for all the pages).
02158             List<(int annotationObjectNum, (double, double, double, double) annotationOrigin, int
      annotationDestinationPage, (double, double, double, double) annotationDestination)>
      postponedAnnotations = new List<(int, (double, double, double, double), int, (double, double, double,
      double))>();
02159 
02160             //Page
02161             for (int i = 0; i < document.Pages.Count; i++)
02162             {
02163                 List<int> annotationsToInclude = new List<int>();
02164 
02165                 //Annotations
02166                 for (int j = 0; j < taggedObjectRectsByPage[i].Count; j++)
02167                 {
02168                     if (linkDestinations.TryGetValue(taggedObjectRectsByPage[i][j].Item1, out string
      destination))
02169                     {
02170                         if (destination.StartsWith("#"))
02171                         {
02172                             //Leave these for later, once we have computed the object number for all
      the pages.  But we need to include the annotation number in the page, so we start processing the
      annotation now.
02173                             for (int k = 0; k < taggedObjectRectsIndicesByPage.Length; k++)
02174                             {
02175                                 if
      (taggedObjectRectsIndicesByPage[k].TryGetValue(destination.Substring(1), out int index))
02176                                 {
02177                                     for (int l = 0; l < taggedObjectRectsByPage[i][j].Item2.Count;
      l++)
02178                                     {
02179                                         objectPositions.Add(position);
02180                                         annotationsToInclude.Add(objectNum);
02181                                         postponedAnnotations.Add((objectNum,
      taggedObjectRectsByPage[i][j].Item2[l], k, taggedObjectRectsByPage[k][index].Item2[0]));
02182                                         objectNum++;
02183                                     }
02184 
02185                                     //Only consider the first match for the local destination.
02186                                     break;
02187                                 }
02188                             }
02189                         }
02190                         else
02191                         {
02192                             for (int l = 0; l < taggedObjectRectsByPage[i][j].Item2.Count; l++)
02193                             {
02194                                 objectPositions.Add(position);
02195                                 currObject = objectNum.ToString() + " 0 obj\n« /Type /Annot /Subtype
      /Link /Rect [" + taggedObjectRectsByPage[i][j].Item2[l].Item1.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      taggedObjectRectsByPage[i][j].Item2[l].Item2.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      taggedObjectRectsByPage[i][j].Item2[l].Item3.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      taggedObjectRectsByPage[i][j].Item2[l].Item4.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + "] " +
02196                                     "/A « /Type /Action /S /URI /URI (" + destination + ") »
      »\nendobj\n";
02197                                 sw.Write(currObject);
02198                                 annotationsToInclude.Add(objectNum);
02199                                 objectNum++;
02200                                 position += currObject.Length;
02201                             }
02202                         }
02203                     }
02204                 }
02205 
02206                 //Page
02207                 objectPositions.Add(position);
02208                 currObject = objectNum.ToString() + " 0 obj\n« /Type /Page /Parent " +
      pageParent.ToString() + " 0 R /MediaBox [0 0 " +
      document.Pages[i].Width.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      document.Pages[i].Height.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + "] /Resources " + resourceObject.ToString() + " 0
      R /Contents " + pageContentInd[i].ToString() + " 0 R ";
02209 
02210                 if (annotationsToInclude.Count > 0)
02211                 {
02212                     StringBuilder annotations = new StringBuilder();
02213                     annotations.Append("/Annots [ ");
```

```
02214                     for (int j = 0; j < annotationsToInclude.Count; j++)
02215                     {
02216                         annotations.Append(annotationsToInclude[j].ToString() + " 0 R ");
02217                     }
02218                     annotations.Append("] ");
02219                     currObject += annotations.ToString();
02220                 }
02221 
02222                 currObject += "»\nendobj\n";
02223 
02224                 sw.Write(currObject);
02225                 pageObjectNums.Add(objectNum);
02226                 objectNum++;
02227                 position += currObject.Length;
02228             }
02229 
02230             //Now we have enough information for the postponed annotations.
02231             foreach ((int annotationObjectNum, (double, double, double, double) annotationOrigin, int
    annotationDestinationPage, (double, double, double, double) annotationDestination) in
    postponedAnnotations)
02232             {
02233                 objectPositions[annotationObjectNum - 1] = position;
02234                 currObject = annotationObjectNum.ToString() + " 0 obj\n« /Type /Annot /Subtype /Link
    /Rect [" + annotationOrigin.Item1.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    annotationOrigin.Item2.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    annotationOrigin.Item3.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    annotationOrigin.Item4.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + "] " +
02235                     "/Dest [ " + pageObjectNums[annotationDestinationPage].ToString() + " 0 R /XYZ " +
    annotationDestination.Item1.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    annotationDestination.Item4.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " 0 ] »\nendobj\n";
02236                 sw.Write(currObject);
02237                 position += currObject.Length;
02238             }
02239 
02240             //Pages
02241             objectPositions[pageParent - 1] = position;
02242             currObject = pageParent.ToString() + " 0 obj\n« /Type /Pages /Kids [ ";
02243             for (int i = 0; i < document.Pages.Count; i++)
02244             {
02245                 //currObject += (pageParent + i + 1).ToString() + " 0 R ";
02246                 currObject += pageObjectNums[i].ToString() + " 0 R ";
02247             }
02248             currObject += "] /Count " + document.Pages.Count + " »\nendobj\n\n";
02249             sw.Write(currObject);
02250             position += currObject.Length;
02251 
02252             //XRef
02253             sw.Write("xref\n0 " + (objectPositions.Count + 1).ToString() + "\n0000000000 65535 f \n");
02254             for (int i = 0; i < objectPositions.Count; i++)
02255             {
02256                 sw.Write(objectPositions[i].ToString("0000000000",
    System.Globalization.CultureInfo.InvariantCulture) + " 00000 n \n");
02257             }
02258 
02259             //Trailer
02260             sw.Write("trailer\n« /Size " + (objectPositions.Count + 1).ToString() + " /Root " +
    rootObject.ToString() + " 0 R »\nstartxref\n" + position.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + "\n%%EOF\n");
02261 
02262             sw.Flush();
02263             sw.Dispose();
02264         }
02265 
02266         private static void WriteGradient(bool includeMatrix, ref GradientBrush gradient, ref
    List<long> objectPositions, ref long position, ref string currObject, ref int objectNum, ref
    StreamWriter sw, ref bool hasAlpha, ref double[,] matrix, ref List<int> gradientIndices)
02267         {
02268             int functionObject = -1;
02269 
02270             if (gradient.GradientStops.Count == 2)
02271             {
02272                 objectPositions.Add(position);
02273 
02274                 currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 2 /Domain [ 0 1 ] /C0 [ "
    + gradient.GradientStops[0].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " "
    + gradient.GradientStops[0].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + " "
    + gradient.GradientStops[0].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ]
    ";
02275                 currObject += "/C1 [ " +
    gradient.GradientStops[1].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    gradient.GradientStops[1].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
```

```
      gradient.GradientStops[1].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ]
      /N 1 »\nendobj\n";
02276                 sw.Write(currObject);
02277                 functionObject = objectNum;
02278
02279                 position += currObject.Length;
02280                 objectNum++;
02281
02282                 hasAlpha = gradient.GradientStops[0].Colour.A != 1 ||
      gradient.GradientStops[1].Colour.A != 1;
02283             }
02284             else
02285             {
02286                 List<double> bounds = new List<double>();
02287                 List<int> functionIndices = new List<int>();
02288
02289                 for (int j = 0; j < gradient.GradientStops.Count - 1; j++)
02290                 {
02291                     objectPositions.Add(position);
02292
02293                     currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 2 /Domain [ 0 1 ] /C0
      [ " + gradient.GradientStops[j].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) +
      " " + gradient.GradientStops[j].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) +
      " " + gradient.GradientStops[j].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture) +
      " ] ";
02294                     currObject += "/C1 [ " + gradient.GradientStops[j +
      1].Colour.R.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      gradient.GradientStops[j + 1].Colour.G.ToString(System.Globalization.CultureInfo.InvariantCulture) + "
      " + gradient.GradientStops[j + 1].Colour.B.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " ] /N 1 »\nendobj\n";
02295                     sw.Write(currObject);
02296                     functionIndices.Add(objectNum);
02297
02298                     if (j < gradient.GradientStops.Count - 2)
02299                     {
02300                         bounds.Add(gradient.GradientStops[j + 1].Offset);
02301                     }
02302
02303                     position += currObject.Length;
02304                     objectNum++;
02305
02306                     if (gradient.GradientStops[j].Colour.A != 1)
02307                     {
02308                         hasAlpha = true;
02309                     }
02310                 }
02311
02312                 if (gradient.GradientStops[gradient.GradientStops.Count - 1].Colour.A != 1)
02313                 {
02314                     hasAlpha = true;
02315                 }
02316
02317                 objectPositions.Add(position);
02318
02319                 currObject = objectNum.ToString() + " 0 obj\n« /FunctionType 3 /Domain [ 0 1 ]
      /Functions [ ";
02320
02321                 for (int j = 0; j < functionIndices.Count; j++)
02322                 {
02323                     currObject +=
      functionIndices[j].ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R ";
02324                 }
02325
02326                 currObject += "] /Bounds [ ";
02327
02328                 for (int j = 0; j < bounds.Count; j++)
02329                 {
02330                     currObject +=
      bounds[j].ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
02331                 }
02332
02333                 currObject += "] /Encode [ ";
02334
02335                 for (int j = 0; j < functionIndices.Count; j++)
02336                 {
02337                     currObject += "0 1 ";
02338                 }
02339
02340                 currObject += "] »\nendobj\n";
02341
02342
02343                 sw.Write(currObject);
02344                 functionObject = objectNum;
02345
02346                 position += currObject.Length;
02347                 objectNum++;
02348
```

```
02349             }
02350
02351             if (gradient is LinearGradientBrush linear)
02352             {
02353                 objectPositions.Add(position);
02354
02355                 currObject = objectNum.ToString() + " 0 obj\n« /Type /Pattern /PatternType 2 ";
02356
02357                 if (includeMatrix)
02358                 {
02359                     currObject += "/Matrix [ " +
02360
02361                     matrix[0, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02362                     matrix[1, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02363                     matrix[0, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02364                     matrix[1, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02365                     matrix[0, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02366                     matrix[1, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ] ";
02367                 }
02368
02369                 currObject += "/Shading « /ShadingType 2 /ColorSpace /DeviceRGB /Coords [ " +
      linear.StartPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.StartPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.EndPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      linear.EndPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ] ";
02370
02371                 currObject += "/Domain [ 0 1 ] /Extend [ true true ] /Function " +
      functionObject.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R » »\nendobj\n";
02372                 sw.Write(currObject);
02373                 gradientIndices.Add(objectNum);
02374
02375                 position += currObject.Length;
02376                 objectNum++;
02377             }
02378             else if (gradient is RadialGradientBrush radial)
02379             {
02380                 objectPositions.Add(position);
02381
02382                 currObject = objectNum.ToString() + " 0 obj\n« /Type /Pattern /PatternType 2 ";
02383
02384                 if (includeMatrix)
02385                 {
02386                     currObject += "/Matrix [ " +
02387
02388                     matrix[0, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02389                     matrix[1, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02390                     matrix[0, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02391                     matrix[1, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02392                     matrix[0, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
02393                     matrix[1, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ] ";
02394                 }
02395
02396                 currObject += "/Shading « /ShadingType 3 /ColorSpace /DeviceRGB /Coords [ " +
      radial.FocalPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.FocalPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 " +
      radial.Centre.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.Centre.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      radial.Radius.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ] ";
02397
02398                 currObject += "/Domain [ 0 1 ] /Extend [ true true ] /Function " +
      functionObject.ToString(System.Globalization.CultureInfo.InvariantCulture) + " 0 R » »\nendobj\n";
02399                 sw.Write(currObject);
02400                 gradientIndices.Add(objectNum);
02401
02402                 position += currObject.Length;
02403                 objectNum++;
02404             }
02405         }
02406
02407         private static (double, double, double, double) MeasureFigure(IFigure figure, ref double[,]
      transformationMatrix, Stack<double[,]> savedStates)
02408         {
02409             if (figure is PathFigure)
02410             {
02411                 PathFigure fig = figure as PathFigure;
```

```
02412
02413                 double minX = double.MaxValue;
02414                 double maxX = double.MinValue;
02415                 double minY = double.MaxValue;
02416                 double maxY = double.MinValue;
02417
02418                 for (int i = 0; i < fig.Segments.Length; i++)
02419                 {
02420                     switch (fig.Segments[i].Type)
02421                     {
02422                         case SegmentType.Move:
02423                             {
02424                                 Point pt = transformationMatrix.Multiply(fig.Segments[i].Point);
02425                                 minX = Math.Min(minX, pt.X);
02426                                 minY = Math.Min(minY, pt.Y);
02427                                 maxX = Math.Max(maxX, pt.X);
02428                                 maxY = Math.Max(maxY, pt.Y);
02429                             }
02430                             break;
02431                         case SegmentType.Line:
02432                             {
02433                                 Point pt = transformationMatrix.Multiply(fig.Segments[i].Point);
02434                                 minX = Math.Min(minX, pt.X);
02435                                 minY = Math.Min(minY, pt.Y);
02436                                 maxX = Math.Max(maxX, pt.X);
02437                                 maxY = Math.Max(maxY, pt.Y);
02438                             }
02439                             break;
02440                         case SegmentType.CubicBezier:
02441                             for (int j = 0; j < fig.Segments[i].Points.Length; j++)
02442                             {
02443                                 Point pt = transformationMatrix.Multiply(fig.Segments[i].Points[j]);
02444                                 minX = Math.Min(minX, pt.X);
02445                                 minY = Math.Min(minY, pt.Y);
02446                                 maxX = Math.Max(maxX, pt.X);
02447                                 maxY = Math.Max(maxY, pt.Y);
02448                             }
02449                             break;
02450                         case SegmentType.Close:
02451                             break;
02452                     }
02453                 }
02454
02455                 return (minX, minY, maxX, maxY);
02456             }
02457             else if (figure is TextFigure)
02458             {
02459                 TextFigure fig = figure as TextFigure;
02460
02461                 double realX = fig.Position.X;
02462
02463                 if (fig.Font.FontFamily.TrueTypeFile != null)
02464                 {
02465                     realX = fig.Position.X -
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(fig.Text[0]).LeftSideBearing *
      fig.Font.FontSize / 1000;
02466                 }
02467
02468                 double yMax = 0;
02469                 double yMin = 0;
02470
02471                 if (fig.Font.FontFamily.TrueTypeFile != null)
02472                 {
02473                     for (int i = 0; i < fig.Text.Length; i++)
02474                     {
02475                         TrueTypeFile.VerticalMetrics vMet =
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(fig.Text[i]);
02476                         yMin = Math.Min(yMin, vMet.YMin * fig.Font.FontSize / 1000);
02477                         yMax = Math.Max(yMax, vMet.YMax * fig.Font.FontSize / 1000);
02478                     }
02479                 }
02480
02481                 double realY = fig.Position.Y;
02482
02483                 if (fig.TextBaseline == TextBaselines.Bottom)
02484                 {
02485                     realY -= yMax;
02486                 }
02487                 else if (fig.TextBaseline == TextBaselines.Top)
02488                 {
02489                     realY -= yMin;
02490                 }
02491                 else if (fig.TextBaseline == TextBaselines.Middle)
02492                 {
02493                     realY -= (yMax + yMin) * 0.5;
02494                 }
02495                 else if (fig.TextBaseline == TextBaselines.Baseline)
```

```
02496                 {
02497                     realY -= yMax + yMin;
02498                 }
02499 
02500                 Font.DetailedFontMetrics metrics = fig.Font.MeasureTextAdvanced(fig.Text);
02501 
02502                 Point corner1 = new Point(fig.Position.X - metrics.LeftSideBearing, realY +
      metrics.Top);
02503                 Point corner2 = new Point(fig.Position.X + metrics.Width, realY + metrics.Top);
02504                 Point corner3 = new Point(fig.Position.X + metrics.Width, realY + metrics.Bottom);
02505                 Point corner4 = new Point(fig.Position.X - metrics.LeftSideBearing, realY +
      metrics.Bottom);
02506 
02507                 corner1 = transformationMatrix.Multiply(corner1);
02508                 corner2 = transformationMatrix.Multiply(corner2);
02509                 corner3 = transformationMatrix.Multiply(corner3);
02510                 corner4 = transformationMatrix.Multiply(corner4);
02511 
02512                 return (Math.Min(corner1.X, Math.Min(corner2.X, Math.Min(corner3.X, corner4.X))),
      Math.Min(corner1.Y, Math.Min(corner2.Y, Math.Min(corner3.Y, corner4.Y))), Math.Max(corner1.X,
      Math.Max(corner2.X, Math.Max(corner3.X, corner4.X))), Math.Max(corner1.Y, Math.Max(corner2.Y,
      Math.Max(corner3.Y, corner4.Y))));
02513             }
02514             else if (figure is TransformFigure transf)
02515             {
02516                 if (transf.TransformType == TransformFigure.TransformTypes.Transform)
02517                 {
02518                     transformationMatrix = transformationMatrix.Multiply(transf.TransformationMatrix);
02519                 }
02520                 else if (transf.TransformType == TransformFigure.TransformTypes.Save)
02521                 {
02522                     savedStates.Push(transformationMatrix);
02523                 }
02524                 else if (transf.TransformType == TransformFigure.TransformTypes.Restore)
02525                 {
02526                     transformationMatrix = savedStates.Pop();
02527                 }
02528 
02529                 return (0, 0, 0, 0);
02530             }
02531             else if (figure is RasterImageFigure)
02532             {
02533                 Point corner1 = new Point(0, 0);
02534                 Point corner2 = new Point(0, 1);
02535                 Point corner3 = new Point(1, 1);
02536                 Point corner4 = new Point(1, 0);
02537 
02538                 corner1 = transformationMatrix.Multiply(corner1);
02539                 corner2 = transformationMatrix.Multiply(corner2);
02540                 corner3 = transformationMatrix.Multiply(corner3);
02541                 corner4 = transformationMatrix.Multiply(corner4);
02542 
02543                 return (Math.Min(corner1.X, Math.Min(corner2.X, Math.Min(corner3.X, corner4.X))),
      Math.Min(corner1.Y, Math.Min(corner2.Y, Math.Min(corner3.Y, corner4.Y))), Math.Max(corner1.X,
      Math.Max(corner2.X, Math.Max(corner3.X, corner4.X))), Math.Max(corner1.Y, Math.Max(corner2.Y,
      Math.Max(corner3.Y, corner4.Y))));
02544             }
02545             else
02546             {
02547                 return (0, 0, 0, 0);
02548             }
02549         }
02550 
02551         private static double[,] Multiply(this double[,] matrix, double[,] matrix2)
02552         {
02553             double[,] tbr = new double[3, 3];
02554 
02555             tbr[0, 0] = matrix[0, 0] * matrix2[0, 0] - matrix[0, 1] * matrix2[1, 0] + matrix[0, 2] *
      matrix2[2, 0];
02556             tbr[0, 1] = -matrix[0, 0] * matrix2[0, 1] + matrix[0, 1] * matrix2[1, 1] + matrix[0, 2] *
      matrix2[2, 1];
02557             tbr[0, 2] = matrix[0, 0] * matrix2[0, 2] + matrix[0, 1] * matrix2[1, 2] + matrix[0, 2] *
      matrix2[2, 2];
02558 
02559             tbr[1, 0] = matrix[1, 0] * matrix2[0, 0] - matrix[1, 1] * matrix2[1, 0] + matrix[1, 2] *
      matrix2[2, 0];
02560             tbr[1, 1] = -matrix[1, 0] * matrix2[0, 1] + matrix[1, 1] * matrix2[1, 1] + matrix[1, 2] *
      matrix2[2, 1];
02561             tbr[1, 2] = matrix[1, 0] * matrix2[0, 2] + matrix[1, 1] * matrix2[1, 2] + matrix[1, 2] *
      matrix2[2, 2];
02562 
02563             tbr[2, 0] = matrix[2, 0] * matrix2[0, 0] - matrix[2, 1] * matrix2[1, 0] + matrix[2, 2] *
      matrix2[2, 0];
02564             tbr[2, 1] = -matrix[2, 0] * matrix2[0, 1] + matrix[2, 1] * matrix2[1, 1] + matrix[2, 2] *
      matrix2[2, 1];
02565             tbr[2, 2] = matrix[2, 0] * matrix2[0, 2] + matrix[2, 1] * matrix2[1, 2] + matrix[2, 2] *
      matrix2[2, 2];
```

```
02566
02567             return tbr;
02568         }
02569
02570         private static Point Multiply(this double[,] matrix, Point point)
02571         {
02572             double[] transPt = new double[3];
02573
02574             transPt[0] = matrix[0, 0] * point.X + matrix[0, 1] * point.Y + matrix[0, 2];
02575             transPt[1] = matrix[1, 0] * point.X + matrix[1, 1] * point.Y + matrix[1, 2];
02576             transPt[2] = matrix[2, 0] * point.X + matrix[2, 1] * point.Y + matrix[2, 2];
02577
02578             return new Point(transPt[0] / transPt[2], transPt[1] / transPt[2]);
02579         }
02580
02581         private static string FigureAsPDFString(IFigure figure, Dictionary<string, string>
      nonSymbolFontIds, Dictionary<string, string> symbolFontIds, Dictionary<string, Dictionary<char, int»
      symbolGlyphIndices, double[] alphas, Dictionary<string, int> imageObjectNums, double[,]
      transformationMatrix, List<(GradientBrush, double[,], IFigure)> gradients)
02582         {
02583
02584             StringBuilder sb = new StringBuilder();
02585
02586             bool restoreAtEnd = false;
02587
02588             if (figure.Fill != null)
02589             {
02590                 if (figure.Fill is SolidColourBrush solid)
02591                 {
02592                     sb.Append(solid.R.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.G.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.B.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " rg\n");
02593                     sb.Append("/a" + Array.IndexOf(alphas, solid.A).ToString() + " gs\n");
02594                 }
02595                 else if (figure.Fill is GradientBrush gradient)
02596                 {
02597                     int brushIndex = gradients.Count;
02598                     double[,] clonedMatrix = new double[3, 3] { { transformationMatrix[0, 0],
      transformationMatrix[0, 1], transformationMatrix[0, 2] }, { transformationMatrix[1, 0],
      transformationMatrix[1, 1], transformationMatrix[1, 2] }, { transformationMatrix[2, 0],
      transformationMatrix[2, 1], transformationMatrix[2, 2] } };
02599
02600                     gradients.Add((gradient, clonedMatrix, figure));
02601
02602                     bool gradientCompatible = PDFContext.IsCompatible(gradient);
02603
02604                     if (!gradientCompatible)
02605                     {
02606                         sb.Append("q\n");
02607                         sb.Append("/ma" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " gs ");
02608                         restoreAtEnd = true;
02609                     }
02610
02611                     sb.Append("/Pattern cs /p" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " scn /a" +
      Array.IndexOf(alphas, 1.0).ToString() + " gs\n");
02612                 }
02613             }
02614
02615             if (figure.Stroke != null)
02616             {
02617                 if (figure.Stroke is SolidColourBrush solid)
02618                 {
02619                     sb.Append(solid.R.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.G.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.B.ToString("0.#################",
      System.Globalization.CultureInfo.InvariantCulture) + " RG\n");
02620                     sb.Append("/a" + Array.IndexOf(alphas, solid.A).ToString() + " gs\n");
02621                 }
02622                 else if (figure.Stroke is GradientBrush gradient)
02623                 {
02624                     int brushIndex = gradients.Count;
02625                     double[,] clonedMatrix = new double[3, 3] { { transformationMatrix[0, 0],
      transformationMatrix[0, 1], transformationMatrix[0, 2] }, { transformationMatrix[1, 0],
      transformationMatrix[1, 1], transformationMatrix[1, 2] }, { transformationMatrix[2, 0],
      transformationMatrix[2, 1], transformationMatrix[2, 2] } };
02626
02627                     gradients.Add((gradient, clonedMatrix, figure));
02628
02629                     bool gradientCompatible = PDFContext.IsCompatible(gradient);
02630
02631                     if (!gradientCompatible)
02632                     {
02633                         sb.Append("q\n");
02634                         sb.Append("/ma" +
```

```
    brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " gs ");
02635                       restoreAtEnd = true;
02636                   }
02637
02638                   sb.Append("/Pattern CS /p" +
    brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " SCN /a" +
    Array.IndexOf(alphas, 1.0).ToString() + " gs\n");
02639               }
02640
02641               sb.Append(figure.LineWidth.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " w\n");
02642               sb.Append(((int)figure.LineCap).ToString() + " J\n");
02643               sb.Append(((int)figure.LineJoin).ToString() + " j\n");
02644               if (figure.LineDash.UnitsOff != 0 || figure.LineDash.UnitsOn != 0)
02645               {
02646                   sb.Append("[ " + figure.LineDash.UnitsOn.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    figure.LineDash.UnitsOff.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " ] " +
    figure.LineDash.Phase.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " d\n");
02647               }
02648               else
02649               {
02650                   sb.Append("[] 0 d\n");
02651               }
02652           }
02653
02654           if (figure is PathFigure)
02655           {
02656               PathFigure fig = figure as PathFigure;
02657
02658               for (int i = 0; i < fig.Segments.Length; i++)
02659               {
02660                   switch (fig.Segments[i].Type)
02661                   {
02662                       case SegmentType.Move:
02663                           {
02664                               sb.Append(fig.Segments[i].Point.X.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    fig.Segments[i].Point.Y.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " m ");
02665                           }
02666                           break;
02667                       case SegmentType.Line:
02668                           {
02669                               sb.Append(fig.Segments[i].Point.X.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    fig.Segments[i].Point.Y.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " l ");
02670                           }
02671                           break;
02672                       case SegmentType.CubicBezier:
02673                           for (int j = 0; j < fig.Segments[i].Points.Length; j++)
02674                           {
02675                               sb.Append(fig.Segments[i].Points[j].X.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " " +
    fig.Segments[i].Points[j].Y.ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture) + " ");
02676                           }
02677                           sb.Append("c ");
02678                           break;
02679                       case SegmentType.Close:
02680                           sb.Append("h ");
02681                           break;
02682                   }
02683               }
02684
02685               if (fig.IsClipping)
02686               {
02687                   sb.Append("W n\n");
02688               }
02689               else
02690               {
02691                   if (fig.Fill != null)
02692                   {
02693                       if (fig.FillRule == FillRule.EvenOdd)
02694                       {
02695                           sb.Append("f*\n");
02696                       }
02697                       else
02698                       {
02699                           sb.Append("f\n");
02700                       }
02701                   }
02702
02703                   if (fig.Stroke != null)
```

```
02704                     {
02705                         sb.Append("S\n");
02706                     }
02707                 }
02708             }
02709             else if (figure is TextFigure)
02710             {
02711                 TextFigure fig = figure as TextFigure;
02712 
02713                 List<(string txt, bool isSymbolic)> segments = new List<(string txt, bool
      isSymbolic)>();
02714 
02715                 StringBuilder currSeg = new StringBuilder();
02716                 bool currSymbolic = false;
02717 
02718                 for (int i = 0; i < fig.Text.Length; i++)
02719                 {
02720                     if (CP1252Chars.Contains(fig.Text[i]))
02721                     {
02722                         if (!currSymbolic)
02723                         {
02724                             currSeg.Append(fig.Text[i]);
02725                         }
02726                         else
02727                         {
02728                             if (currSeg.Length > 0)
02729                             {
02730                                 segments.Add((currSeg.ToString(), currSymbolic));
02731                             }
02732 
02733                             currSeg = new StringBuilder();
02734                             currSymbolic = false;
02735                             currSeg.Append(fig.Text[i]);
02736                         }
02737                     }
02738                     else
02739                     {
02740                         if (currSymbolic)
02741                         {
02742                             currSeg.Append(fig.Text[i]);
02743                         }
02744                         else
02745                         {
02746                             if (currSeg.Length > 0)
02747                             {
02748                                 segments.Add((currSeg.ToString(), currSymbolic));
02749                             }
02750 
02751                             currSeg = new StringBuilder();
02752                             currSymbolic = true;
02753                             currSeg.Append(fig.Text[i]);
02754                         }
02755                     }
02756                 }
02757 
02758                 if (currSeg.Length > 0)
02759                 {
02760                     segments.Add((currSeg.ToString(), currSymbolic));
02761                 }
02762 
02763 
02764 
02765                 double realX = fig.Position.X;
02766 
02767                 if (fig.Font.FontFamily.TrueTypeFile != null)
02768                 {
02769                     realX = fig.Position.X -
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(fig.Text[0]).LeftSideBearing *
      fig.Font.FontSize / 1000;
02770                 }
02771 
02772                 double yMax = 0;
02773                 double yMin = 0;
02774 
02775                 if (fig.Font.FontFamily.TrueTypeFile != null)
02776                 {
02777                     for (int i = 0; i < fig.Text.Length; i++)
02778                     {
02779                         TrueTypeFile.VerticalMetrics vMet =
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(fig.Text[i]);
02780                         yMin = Math.Min(yMin, vMet.YMin * fig.Font.FontSize / 1000);
02781                         yMax = Math.Max(yMax, vMet.YMax * fig.Font.FontSize / 1000);
02782                     }
02783                 }
02784 
02785                 double realY = fig.Position.Y;
02786 
```

```
02787                 if (fig.TextBaseline == TextBaselines.Bottom)
02788                 {
02789                     realY -= yMax;
02790                 }
02791                 else if (fig.TextBaseline == TextBaselines.Top)
02792                 {
02793                     realY -= yMin;
02794                 }
02795                 else if (fig.TextBaseline == TextBaselines.Middle)
02796                 {
02797                     realY -= (yMax + yMin) * 0.5;
02798                 }
02799                 else if (fig.TextBaseline == TextBaselines.Baseline)
02800                 {
02801                     realY -= yMax + yMin;
02802                 }
02803 
02804                 double middleY = realY + (yMax + yMin) * 0.5;
02805 
02806 
02807 
02808                 sb.Append("q\n1 0 0 1 0 " + (middleY).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
02809                 sb.Append("1 0 0 -1 0 0 cm\n");
02810                 sb.Append("1 0 0 1 0 " + (-middleY).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
02811 
02812                 sb.Append("BT\n");
02813 
02814                 if (figure.Stroke != null && figure.Fill != null)
02815                 {
02816                     sb.Append("2 Tr\n");
02817                 }
02818                 else if (figure.Stroke != null)
02819                 {
02820                     sb.Append("1 Tr\n");
02821                 }
02822                 else if (figure.Fill != null)
02823                 {
02824                     sb.Append("0 Tr\n");
02825                 }
02826 
02827                 sb.Append(realX.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + realY.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " Td\n");
02828 
02829                 for (int i = 0; i < segments.Count; i++)
02830                 {
02831                     if (!segments[i].isSymbolic)
02832                     {
02833                         sb.Append("/" + nonSymbolFontIds[fig.Font.FontFamily.FamilyName] + " " +
      fig.Font.FontSize.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) +
      " Tf\n");
02834                         sb.Append(GetKernedString(segments[i].txt, fig.Font) + " TJ\n");
02835                     }
02836                     else
02837                     {
02838                         sb.Append("/" + symbolFontIds[fig.Font.FontFamily.FamilyName] + " " +
      fig.Font.FontSize.ToString("0.################", System.Globalization.CultureInfo.InvariantCulture) +
      " Tf\n");
02839                         sb.Append("<" + EscapeSymbolStringForPDF(segments[i].txt,
      symbolGlyphIndices[fig.Font.FontFamily.FamilyName]) + "> Tj\n");
02840                     }
02841                 }
02842 
02843                 sb.Append("ET\nQ\n");
02844             }
02845             else if (figure is TransformFigure transf)
02846             {
02847                 if (transf.TransformType == TransformFigure.TransformTypes.Transform)
02848                 {
02849                     sb.Append(transf.TransformationMatrix[0, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
02850                     sb.Append(transf.TransformationMatrix[0, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
02851                     sb.Append(transf.TransformationMatrix[1, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
02852                     sb.Append(transf.TransformationMatrix[1, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
02853                     sb.Append(transf.TransformationMatrix[0, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
02854                     sb.Append(transf.TransformationMatrix[1, 2].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
02855                 }
02856                 else if (transf.TransformType == TransformFigure.TransformTypes.Save)
02857                 {
02858                     sb.Append("q\n");
```

```
02859                 }
02860                 else if (transf.TransformType == TransformFigure.TransformTypes.Restore)
02861                 {
02862                     sb.Append("Q\n");
02863                 }
02864             }
02865             else if (figure is RasterImageFigure fig)
02866             {
02867                 sb.Append("/a" + Array.IndexOf(alphas, 1).ToString() + " gs\n");
02868 
02869                 int imageNum = imageObjectNums[fig.Image.Id];
02870 
02871                 sb.Append("/Img" + imageNum.ToString() + " Do\n");
02872             }
02873 
02874             if (restoreAtEnd)
02875             {
02876                 sb.Append("Q\n");
02877             }
02878 
02879             return sb.ToString();
02880         }
02881 
02882         internal static MemoryStream ZLibCompress(Stream contentStream)
02883         {
02884             MemoryStream compressedStream = new MemoryStream();
02885             compressedStream.Write(new byte[] { 0x78, 0x01 }, 0, 2);
02886 
02887             using (DeflateStream deflate = new DeflateStream(compressedStream,
      CompressionLevel.Optimal, true))
02888             {
02889                 contentStream.CopyTo(deflate);
02890             }
02891             contentStream.Seek(0, SeekOrigin.Begin);
02892 
02893             uint checksum = Adler32(contentStream);
02894 
02895             compressedStream.Write(new byte[] { (byte)((checksum » 24) & 255), (byte)((checksum » 16)
      & 255), (byte)((checksum » 8) & 255), (byte)(checksum & 255) }, 0, 4);
02896 
02897             compressedStream.Seek(0, SeekOrigin.Begin);
02898 
02899             return compressedStream;
02900         }
02901 
02902         internal static uint Adler32(Stream contentStream)
02903         {
02904             uint s1 = 1;
02905             uint s2 = 0;
02906 
02907             int readByte;
02908 
02909             while ((readByte = contentStream.ReadByte()) >= 0)
02910             {
02911                 s1 = (s1 + (byte)readByte) % 65521U;
02912                 s2 = (s2 + s1) % 65521U;
02913             }
02914 
02915             return (s2 « 16) + s1;
02916         }
02917     }
02918 }
```

8.21 Axes.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
```

```
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// A plot element that draws an axis line and an arrow.
00025 /// </summary>
00026     public class ContinuousAxis :  IPlotElement
00027     {
00028 /// <summary>
00029 /// The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space
      coordinates.
00030 /// </summary>
00031         public IReadOnlyList<double> StartPoint { get; set; }
00032
00033 /// <summary>
00034 /// The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space
      coordinates.
00035 /// </summary>
00036         public IReadOnlyList<double> EndPoint { get; set; }
00037
00038 /// <summary>
00039 /// The size of the arrow at the end of the axis, expressed in plot space coordinates.
00040 /// </summary>
00041         public double ArrowSize { get; set; } = 3;
00042
00043 /// <summary>
00044 /// The coordinate system used to transform the points from data space to plot space.
00045 /// </summary>
00046         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00047         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00048
00049 /// <summary>
00050 /// Presentation attributes determining the appearance of the axis.
00051 /// </summary>
00052         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00053
00054 /// <summary>
00055 /// A tag to identify the axis in the plot.
00056 /// </summary>
00057         public string Tag { get; set; }
00058
00059 /// <summary>
00060 /// Creates a new <see cref="ContinuousAxis"/> instance.
00061 /// </summary>
00062 /// <param name="startPoint">The starting point of the axis (i.e., the blunt end of the arrow),
      expressed in data space coordinates.</param>
00063 /// <param name="endPoint">The ending point of the axis (i.e., the pointy end of the arrow), expressed
      in data space coordinates.</param>
00064 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00065         public ContinuousAxis(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00066         {
00067             this.StartPoint = startPoint;
00068             this.EndPoint = endPoint;
00069             this.CoordinateSystem = coordinateSystem;
00070         }
00071
00072 /// <inheritdoc/>
00073         public void Plot(Graphics target)
00074         {
00075             Point startPoint = CoordinateSystem.ToPlotCoordinates(StartPoint);
00076             Point endPoint = CoordinateSystem.ToPlotCoordinates(EndPoint);
00077
00078             double[] direction = new double[StartPoint.Count];
00079
00080             for (int i = 0; i < direction.Length; i++)
00081             {
00082                 direction[i] = EndPoint[i] - StartPoint[i];
00083             }
00084
00085             double angle;
00086
00087             GraphicsPath path;
00088
00089             if (CoordinateSystem.IsLinear || CoordinateSystem.IsDirectionStraight(direction))
00090             {
00091                 path = new GraphicsPath().MoveTo(startPoint).LineTo(endPoint);
00092                 angle = Math.Atan2(endPoint.Y - startPoint.Y, endPoint.X - startPoint.X);
00093             }
00094             else
00095             {
00096                 angle = 0;
00097
00098                 path = new GraphicsPath();
```

```
00099
00100                 int count = 0;
00101
00102                 for (int i = 0; i < direction.Length; i++)
00103                 {
00104                     count = Math.Max(count, (int)Math.Ceiling(direction[i] /
      CoordinateSystem.Resolution[i]));
00105                 }
00106
00107                 Point lastPoint = new Point();
00108
00109                 for (int i = 0; i <= count; i++)
00110                 {
00111                     double[] pt = new double[StartPoint.Count];
00112
00113                     for (int j = 0; j < pt.Length; j++)
00114                     {
00115                         pt[j] = StartPoint[j] + direction[j] / count * i;
00116                     }
00117
00118                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00119                     path.LineTo(point);
00120
00121                     if (i == count)
00122                     {
00123                         angle = Math.Atan2(point.Y - lastPoint.Y, point.X - lastPoint.X);
00124                     }
00125                     else
00126                     {
00127                         lastPoint = point;
00128                     }
00129                 }
00130             }
00131
00132             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash, tag:
      Tag);
00133
00134             if (ArrowSize > 0)
00135             {
00136                 target.Save();
00137                 target.Translate(endPoint);
00138
00139                 target.Rotate(angle);
00140
00141                 GraphicsPath arrowPath = new GraphicsPath().MoveTo(-ArrowSize *
      PresentationAttributes.LineWidth, -ArrowSize * PresentationAttributes.LineWidth).LineTo(ArrowSize *
      PresentationAttributes.LineWidth, 0).LineTo(-ArrowSize * PresentationAttributes.LineWidth, ArrowSize *
      PresentationAttributes.LineWidth).Close();
00142
00143                 string fillTag = Tag;
00144                 string strokeTag = Tag;
00145
00146                 if (target.UseUniqueTags && !string.IsNullOrEmpty(Tag))
00147                 {
00148                     fillTag += "@arrowFill";
00149                     strokeTag += "@arrowStroke";
00150                 }
00151
00152                 if (PresentationAttributes.Fill != null)
00153                 {
00154                     target.FillPath(arrowPath, PresentationAttributes.Fill, tag:  fillTag);
00155                 }
00156
00157                 target.StrokePath(arrowPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  strokeTag);
00158
00159                 target.Restore();
00160             }
00161         }
00162     }
00163
00164 /// <summary>
00165 /// A plot element that draws equally spaced ticks on an axis.
00166 /// </summary>
00167     public class ContinuousAxisTicks :  IPlotElement
00168     {
00169 /// <summary>
00170 /// The starting point of the axis, expressed in data space coordinates.
00171 /// </summary>
00172         public IReadOnlyList<double> StartPoint { get; set; }
00173
00174 /// <summary>
00175 /// The ending point of the axis, expressed in data space coordinates.
00176 /// </summary>
00177         public IReadOnlyList<double> EndPoint { get; set; }
```

```
00178
00179 /// <summary>
00180 /// The number of intervals between ticks.  Note that the number of ticks will be one greater than
      this.
00181 /// </summary>
00182         public double IntervalCount { get; set; } = 10;
00183
00184 /// <summary>
00185 /// The size of the ticks "above" the axis.  What "above" means depends on the orientation of the
      axis.
00186 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the tick, and
00187 /// return a <see langword="double"/> representing the size of the tick in plot coordinates.
00188 /// </summary>
00189         public Func<int, double> SizeAbove { get; set; } = i => i % 2 == 0 ?  3 :  2;
00190
00191 /// <summary>
00192 /// The size of the ticks "below" the axis.  What "below" means depends on the orientation of the
      axis.
00193 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the tick, and
00194 /// return a <see langword="double"/> representing the size of the tick in plot coordinates.
00195 /// </summary>
00196         public Func<int, double> SizeBelow { get; set; } = i => i % 2 == 0 ?  3 :  2;
00197
00198 /// <summary>
00199 /// The coordinate system used to transform the points from data space to plot space.
00200 /// </summary>
00201         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00202         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00203
00204 /// <summary>
00205 /// Presentation attributes determining the appearance of the ticks.
00206 /// </summary>
00207         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00208
00209 /// <summary>
00210 /// A tag to identify the ticks in the plot.
00211 /// </summary>
00212         public string Tag { get; set; }
00213
00214 /// <summary>
00215 /// Creates a new <see cref="ContinuousAxisTicks"/> instance.
00216 /// </summary>
00217 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00218 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00219 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00220         public ContinuousAxisTicks(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00221         {
00222             this.StartPoint = startPoint;
00223             this.EndPoint = endPoint;
00224             this.CoordinateSystem = coordinateSystem;
00225         }
00226
00227 /// <inheritdoc/>
00228         public void Plot(Graphics target)
00229         {
00230             double[] direction = new double[StartPoint.Count];
00231             double directionMod = 0;
00232
00233             for (int i = 0; i < direction.Length; i++)
00234             {
00235                 direction[i] = EndPoint[i] - StartPoint[i];
00236                 directionMod += direction[i] * direction[i];
00237             }
00238
00239             directionMod = Math.Sqrt(directionMod);
00240
00241             GraphicsPath tickPath = new GraphicsPath();
00242
00243             double[] normDir = new double[StartPoint.Count];
00244             double[] invDir = new double[StartPoint.Count];
00245
00246             for (int i = 0; i < direction.Length; i++)
00247             {
00248                 normDir[i] = direction[i] / directionMod;
00249                 invDir[i] = -normDir[i];
00250             }
00251
00252             for (int i = 0; i <= IntervalCount; i++)
00253             {
00254                 double[] pt = new double[StartPoint.Count];
00255
```

```
00256                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
      inv.IsDirectionStraight(direction))
00257                 {
00258                     Point start = inv.ToPlotCoordinates(StartPoint);
00259                     Point end = inv.ToPlotCoordinates(EndPoint);
00260 
00261                     pt = inv.ToDataCoordinates(new Point(start.X * (1 - i / IntervalCount) + end.X * i
      / IntervalCount, start.Y * (1 - i / IntervalCount) + end.Y * i / IntervalCount));
00262                 }
00263                 else
00264                 {
00265                     for (int j = 0; j < pt.Length; j++)
00266                     {
00267                         pt[j] = StartPoint[j] + direction[j] / IntervalCount * i;
00268                     }
00269                 }
00270 
00271 
00272                 double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00273                 double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00274 
00275                 Point point = CoordinateSystem.ToPlotCoordinates(pt);
00276                 Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00277                 Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00278 
00279                 Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00280                 double derivMod = deriv.Modulus();
00281                 deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00282                 Point perp = new Point(-deriv.Y, deriv.X);
00283 
00284                 double sizeAbove = SizeAbove(i);
00285                 double sizeBelow = SizeBelow(i);
00286 
00287                 if (sizeAbove + sizeBelow > 0)
00288                 {
00289                     tickPath.MoveTo(point.X - sizeAbove * perp.X, point.Y - sizeAbove * perp.Y);
00290                     tickPath.LineTo(point.X + sizeBelow * perp.X, point.Y + sizeBelow * perp.Y);
00291                 }
00292             }
00293 
00294             target.StrokePath(tickPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  Tag);
00295         }
00296 
00297     }
00298 
00299 /// <summary>
00300 /// A plot element that draws equally spaced labels on an axis.
00301 /// </summary>
00302     public class ContinuousAxisLabels :  IPlotElement
00303     {
00304 /// <summary>
00305 /// The starting point of the axis, expressed in data space coordinates.
00306 /// </summary>
00307         public IReadOnlyList<double> StartPoint { get; set; }
00308 
00309 /// <summary>
00310 /// The ending point of the axis, expressed in data space coordinates.
00311 /// </summary>
00312         public IReadOnlyList<double> EndPoint { get; set; }
00313 
00314 /// <summary>
00315 /// The number of intervals between labels.  Note that the number of labels will be one greater than
      this.
00316 /// </summary>
00317         public double IntervalCount { get; set; } = 10;
00318 
00319 /// <summary>
00320 /// The distance of the label anchor from the point on the axis.
00321 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the label, and
00322 /// return a <see langword="double"/> representing the distance of the label from the axis, in plot
      space coordinates.
00323 /// </summary>
00324         public Func<int, double> Position { get; set; } = _ => 10;
00325 
00326 /// <summary>
00327 /// A function used to determine what text to draw at each label.  You should set this to a
00328 /// function accepting two parameters:  an <see cref="IReadOnlyList{T}"/> of <see
      langword="double"/>s,
00329 /// representing the coordinates of the point the label refers to, and an <see langword="int"/>
00330 /// representing the index of the label.  The function should return an <see cref="IEnumerable{T}" />
      of
00331 /// <see cref="FormattedText" /> objects, representing the text that will be drawn.
00332 /// </summary>
00333         public Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» TextFormat { get; set; }
```

```
00334
00335 /// <summary>
00336 /// The orientation of the label with respect to the horizontal.  If this is <see langword="null"/>,
      the
00337 /// labels will be perpendicular to the axis.
00338 /// </summary>
00339         public double?  Rotation { get; set; } = null;
00340
00341         private IEnumerable<FormattedText> DefaultTextFormat(IReadOnlyList<double> point)
00342         {
00343             double[] direction = new double[StartPoint.Count];
00344             double directionMod = 0;
00345
00346             double maxDirection = 0;
00347             int maxDirectionIndex = 0;
00348
00349             for (int i = 0; i < direction.Length; i++)
00350             {
00351                 direction[i] = EndPoint[i] - StartPoint[i];
00352                 directionMod += direction[i] * direction[i];
00353
00354                 if (Math.Abs(direction[i]) > maxDirection)
00355                 {
00356                     maxDirection = Math.Abs(direction[i]);
00357                     maxDirectionIndex = i;
00358                 }
00359             }
00360
00361             directionMod = Math.Sqrt(directionMod);
00362
00363             if (maxDirection / directionMod >= 0.9)
00364             {
00365                 double range = Math.Abs((EndPoint[maxDirectionIndex] - StartPoint[maxDirectionIndex])
      / IntervalCount);
00366
00367                 string formatString;
00368
00369                 if (range >= 10)
00370                 {
00371                     formatString = "0";
00372                 }
00373                 else if (range >= 1)
00374                 {
00375                     formatString = "0.0";
00376                 }
00377                 else
00378                 {
00379                     formatString = "0." + new string('0', -(int)Math.Floor(Math.Log10(range)) + 1);
00380                 }
00381
00382                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00383                 {
00384                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00385
00386                     return FormattedText.Format(point[maxDirectionIndex].ToString(formatString,
      System.Globalization.CultureInfo.InvariantCulture), (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00387                 }
00388                 else
00389                 {
00390                     return FormattedText.Format(point[maxDirectionIndex].ToString(formatString,
      System.Globalization.CultureInfo.InvariantCulture), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00391                 }
00392             }
00393             else
00394             {
00395                 double pointMod = 0;
00396
00397                 for (int i = 0; i < point.Count; i++)
00398                 {
00399                     pointMod += (point[i] - StartPoint[i]) * (point[i] - StartPoint[i]);
00400                 }
00401
00402                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00403                 {
00404                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00405
00406                     return FormattedText.Format((pointMod / directionMod).ToString("0%",
      System.Globalization.CultureInfo.InvariantCulture), (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00407                 }
00408                 else
00409                 {
00410                     return FormattedText.Format((pointMod / directionMod).ToString("0%",
```

```
      System.Globalization.CultureInfo.InvariantCulture), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00411                 }
00412             }
00413         }
00414
00415 /// <summary>
00416 /// The baseline for the label anchor.
00417 /// </summary>
00418         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00419
00420 /// <summary>
00421 /// The alignment for the label anchor.
00422 /// </summary>
00423         public TextAnchors Alignment { get; set; } = TextAnchors.Left;
00424
00425 /// <summary>
00426 /// The coordinate system used to transform the points from data space to plot space.
00427 /// </summary>
00428         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00429         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00430
00431 /// <summary>
00432 /// Presentation attributes determining the appearance of the labels.
00433 /// </summary>
00434         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00435
00436 /// <summary>
00437 /// A tag to identify the labels in the plot.
00438 /// </summary>
00439         public string Tag { get; set; }
00440
00441 /// <summary>
00442 /// Create a new <see cref="ContinuousAxisLabels"/> instance.
00443 /// </summary>
00444 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00445 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00446 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00447         public ContinuousAxisLabels(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00448         {
00449             this.StartPoint = startPoint;
00450             this.EndPoint = endPoint;
00451             this.CoordinateSystem = coordinateSystem;
00452             this.TextFormat = (p, i) => this.DefaultTextFormat(p);
00453         }
00454
00455 /// <inheritdoc/>
00456         public void Plot(Graphics target)
00457         {
00458             double[] direction = new double[StartPoint.Count];
00459             double directionMod = 0;
00460
00461             for (int i = 0; i < direction.Length; i++)
00462             {
00463                 direction[i] = EndPoint[i] - StartPoint[i];
00464                 directionMod += direction[i] * direction[i];
00465             }
00466
00467             directionMod = Math.Sqrt(directionMod);
00468
00469             double[] normDir = new double[StartPoint.Count];
00470             double[] invDir = new double[StartPoint.Count];
00471
00472             for (int i = 0; i < direction.Length; i++)
00473             {
00474                 normDir[i] = direction[i] / directionMod;
00475                 invDir[i] = -normDir[i];
00476             }
00477
00478             for (int i = 0; i <= IntervalCount; i++)
00479             {
00480                 double[] pt = new double[StartPoint.Count];
00481
00482                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
      inv.IsDirectionStraight(direction))
00483                 {
00484                     Point start = inv.ToPlotCoordinates(StartPoint);
00485                     Point end = inv.ToPlotCoordinates(EndPoint);
00486
00487                     pt = inv.ToDataCoordinates(new Point(start.X * (1 - i / IntervalCount) + end.X * i
      / IntervalCount, start.Y * (1 - i / IntervalCount) + end.Y * i / IntervalCount));
00488                 }
00489                 else
```

```
00490                 {
00491                     for (int j = 0; j < pt.Length; j++)
00492                     {
00493                         pt[j] = StartPoint[j] + direction[j] / IntervalCount * i;
00494                     }
00495                 }
00496 
00497 
00498                 double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00499                 double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00500 
00501                 IEnumerable<FormattedText> text = TextFormat(pt, i);
00502 
00503                 if (text != null)
00504                 {
00505                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00506                     Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00507                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00508 
00509                     Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00510                     double derivMod = deriv.Modulus();
00511                     deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00512                     Point perp = new Point(-deriv.Y, deriv.X);
00513 
00514                     double position = Position(i);
00515 
00516                     target.Save();
00517                     target.Translate(point.X + position * perp.X, point.Y + position * perp.Y);
00518 
00519                     if (Rotation == null)
00520                     {
00521                         target.Rotate(Math.Atan2(perp.Y, perp.X));
00522                     }
00523                     else
00524                     {
00525                         target.Rotate(Rotation ??  0);
00526                     }
00527 
00528 
00529                     double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
      -text.Measure().Width :  -text.Measure().Width * 0.5;
00530 
00531                     string fillTag = Tag;
00532                     string strokeTag = Tag;
00533 
00534                     if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00535                     {
00536                         fillTag = fillTag + "@" + i;
00537                         strokeTag = strokeTag + "@stroke" + i;
00538                     }
00539 
00540                     target.FillText(x, 0, text, PresentationAttributes.Fill, Baseline, fillTag);
00541 
00542                     if (PresentationAttributes.Stroke != null)
00543                     {
00544                         target.StrokeText(x, 0, text, PresentationAttributes.Stroke, Baseline,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00545                     }
00546 
00547                     target.Restore();
00548                 }
00549             }
00550         }
00551     }
00552 
00553 /// <summary>
00554 /// A plot element that draws a title for an axis.
00555 /// </summary>
00556     public class ContinuousAxisTitle :  IPlotElement
00557     {
00558 /// <summary>
00559 /// The starting point of the axis, expressed in data space coordinates.
00560 /// </summary>
00561         public IReadOnlyList<double> StartPoint { get; set; }
00562 
00563 /// <summary>
00564 /// The ending point of the axis, expressed in data space coordinates.
00565 /// </summary>
00566         public IReadOnlyList<double> EndPoint { get; set; }
00567 
00568 /// <summary>
00569 /// The distance between the title and the axis, in plot space coordinates.
00570 /// </summary>
00571         public double Position { get; set; } = 30;
00572 
00573 /// <summary>
```

```
00574 /// The axis title to draw.
00575 /// </summary>
00576         public IEnumerable<FormattedText> Title { get; set; }
00577 
00578 /// <summary>
00579 /// If the axis is not a straight line (e.g., because the coordinate system is not linear),
00580 /// if this is <see langword="true"/> the title will follow the shape of the axis.  If this
00581 /// is <see langword="false"/>, the title will always be drawn on a straight line.
00582 /// </summary>
00583         public bool FollowAxis { get; set; } = true;
00584 
00585 /// <summary>
00586 /// Orientation of the title with respect to the horizontal.  If this is <see langword="null"/>,
00587 /// the title is parallel to the axis.
00588 /// </summary>
00589         public double?  Rotation { get; set; } = null;
00590 
00591 /// <summary>
00592 /// The baseline for the title anchor.
00593 /// </summary>
00594         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00595 
00596 /// <summary>
00597 /// The alignment for the title anchor.
00598 /// </summary>
00599         public TextAnchors Alignment { get; set; } = TextAnchors.Left;
00600 
00601 /// <summary>
00602 /// The coordinate system used to transform the points from data space to plot space.
00603 /// </summary>
00604         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00605         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00606 
00607 /// <summary>
00608 /// Presentation attributes determining the appearance of the title.
00609 /// </summary>
00610         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Font = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke = null,
      Fill = Colours.Black };
00611 
00612 /// <summary>
00613 /// A tag to identify the labels in the plot.
00614 /// </summary>
00615         public string Tag { get; set; }
00616 
00617 /// <summary>
00618 /// Create a new <see cref="ContinuousAxisTitle"/> instance.
00619 /// </summary>
00620 /// <param name="title">The title to draw on the axis.</param>
00621 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00622 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00623 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00624         public ContinuousAxisTitle(IEnumerable<FormattedText> title, IReadOnlyList<double> startPoint,
      IReadOnlyList<double> endPoint, IContinuousCoordinateSystem coordinateSystem)
00625         {
00626             this.StartPoint = startPoint;
00627             this.EndPoint = endPoint;
00628             this.CoordinateSystem = coordinateSystem;
00629             this.Title = title;
00630         }
00631 
00632 /// <summary>
00633 /// Create a new <see cref="ContinuousAxisTitle"/> instance.
00634 /// </summary>
00635 /// <param name="title">The title to draw on the axis.</param>
00636 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00637 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00638 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00639 /// <param name="presentationAttributes">Presentation attributes determining the appearance of the
      title.</param>
00640         public ContinuousAxisTitle(string title, IReadOnlyList<double> startPoint,
      IReadOnlyList<double> endPoint, IContinuousCoordinateSystem coordinateSystem,
      PlotElementPresentationAttributes presentationAttributes = null)
00641         {
00642             this.StartPoint = startPoint;
00643             this.EndPoint = endPoint;
00644             this.CoordinateSystem = coordinateSystem;
00645 
00646             if (presentationAttributes != null)
00647             {
00648                 this.PresentationAttributes = presentationAttributes;
00649             }
```

```
00650
00651             if (title != null)
00652             {
00653                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00654                 {
00655                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00656
00657                     this.Title = FormattedText.Format(title, (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00658                 }
00659                 else
00660                 {
00661                     this.Title = FormattedText.Format(title, PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00662                 }
00663             }
00664             else
00665             {
00666                 Title = null;
00667             }
00668         }
00669
00670 /// <inheritdoc/>
00671         public void Plot(Graphics target)
00672         {
00673             if (Title != null)
00674             {
00675                 double[] direction = new double[StartPoint.Count];
00676                 double directionMod = 0;
00677
00678                 for (int i = 0; i < direction.Length; i++)
00679                 {
00680                     direction[i] = EndPoint[i] - StartPoint[i];
00681                     directionMod += direction[i] * direction[i];
00682                 }
00683
00684                 if (!FollowAxis || CoordinateSystem.IsLinear ||
      CoordinateSystem.IsDirectionStraight(direction))
00685                 {
00686                     directionMod = Math.Sqrt(directionMod);
00687
00688                     double[] normDir = new double[StartPoint.Count];
00689                     double[] invDir = new double[StartPoint.Count];
00690
00691                     for (int i = 0; i < direction.Length; i++)
00692                     {
00693                         normDir[i] = direction[i] / directionMod;
00694                         invDir[i] = -normDir[i];
00695                     }
00696
00697                     double[] pt = new double[StartPoint.Count];
00698
00699                     if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
      inv.IsDirectionStraight(direction))
00700                     {
00701                         Point start = inv.ToPlotCoordinates(StartPoint);
00702                         Point end = inv.ToPlotCoordinates(EndPoint);
00703
00704                         pt = inv.ToDataCoordinates(new Point(start.X * 0.5 + end.X * 0.5, start.Y *
      0.5 + end.Y * 0.5));
00705                     }
00706                     else
00707                     {
00708                         for (int j = 0; j < pt.Length; j++)
00709                         {
00710                             pt[j] = StartPoint[j] + direction[j] * 0.5;
00711                         }
00712                     }
00713
00714                     double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00715                     double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00716
00717                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00718                     Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00719                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00720
00721                     Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00722                     double derivMod = deriv.Modulus();
00723                     deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00724                     Point perp = new Point(-deriv.Y, deriv.X);
00725
00726                     target.Save();
00727                     target.Translate(point.X + Position * perp.X, point.Y + Position * perp.Y);
00728
00729                     if (Rotation == null)
00730                     {
```

```
00731                         target.Rotate(Math.Atan2(perp.Y, perp.X) - Math.PI / 2);
00732                     }
00733                     else
00734                     {
00735                         target.Rotate((Rotation ??  0) - Math.PI / 2);
00736                     }
00737 
00738 
00739                     double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
      -Title.Measure().Width :  -Title.Measure().Width * 0.5;
00740 
00741                     string fillTag = Tag;
00742                     string strokeTag = Tag;
00743 
00744                     if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00745                     {
00746                         strokeTag = strokeTag + "@stroke";
00747                     }
00748 
00749                     target.FillText(x, 0, Title, PresentationAttributes.Fill, Baseline, fillTag);
00750 
00751                     if (PresentationAttributes.Stroke != null)
00752                     {
00753                         target.StrokeText(x, 0, Title, PresentationAttributes.Stroke, Baseline,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00754                     }
00755 
00756                     target.Restore();
00757                 }
00758                 else
00759                 {
00760                     GraphicsPath path;
00761 
00762                     path = new GraphicsPath();
00763 
00764                     int count = 0;
00765 
00766                     for (int i = 0; i < direction.Length; i++)
00767                     {
00768                         count = Math.Max(count, (int)Math.Ceiling(direction[i] /
      CoordinateSystem.Resolution[i]));
00769                     }
00770 
00771                     double[] normDir = new double[StartPoint.Count];
00772                     double[] invDir = new double[StartPoint.Count];
00773 
00774                     for (int i = 0; i < direction.Length; i++)
00775                     {
00776                         normDir[i] = direction[i] / directionMod;
00777                         invDir[i] = -normDir[i];
00778                     }
00779 
00780                     for (int i = 0; i <= count; i++)
00781                     {
00782                         double[] pt = new double[StartPoint.Count];
00783 
00784                         for (int j = 0; j < pt.Length; j++)
00785                         {
00786                             pt[j] = StartPoint[j] + direction[j] / count * i;
00787                         }
00788 
00789                         double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00790                         double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00791 
00792 
00793                         Point point = CoordinateSystem.ToPlotCoordinates(pt);
00794                         Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00795                         Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00796 
00797                         Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00798                         double derivMod = deriv.Modulus();
00799                         deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00800                         Point perp = new Point(-deriv.Y, deriv.X);
00801 
00802                         path.LineTo(point.X + perp.X * Position, point.Y + perp.Y * Position);
00803                     }
00804 
00805                     string fillTag = Tag;
00806                     string strokeTag = Tag;
00807 
00808                     if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00809                     {
00810                         strokeTag = strokeTag + "@stroke";
00811                     }
00812 
00813                     target.FillTextOnPath(path, Title.GetText(), PresentationAttributes.Font,
```

```
      PresentationAttributes.Fill, 0.5, Alignment, Baseline, fillTag);
00814
00815                     if (PresentationAttributes.Stroke != null)
00816                     {
00817                         target.StrokeTextOnPath(path, Title.GetText(), PresentationAttributes.Font,
      PresentationAttributes.Fill, 0.5, Alignment, Baseline, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      strokeTag);
00818                     }
00819                 }
00820             }
00821         }
00822     }
00823
00824 /// <summary>
00825 /// A plot element that draws a grid.
00826 /// </summary>
00827     public class Grid :  IPlotElement
00828     {
00829 /// <summary>
00830 /// The starting point for the first side of the grid.
00831 /// </summary>
00832         public IReadOnlyList<double> Side1Start { get; set; }
00833
00834 /// <summary>
00835 /// The ending point for the first side of the grid.
00836 /// </summary>
00837         public IReadOnlyList<double> Side1End { get; set; }
00838
00839 /// <summary>
00840 /// The starting point for the second side of the grid.
00841 /// </summary>
00842         public IReadOnlyList<double> Side2Start { get; set; }
00843
00844 /// <summary>
00845 /// The ending point for the second side of the grid.
00846 /// </summary>
00847         public IReadOnlyList<double> Side2End { get; set; }
00848
00849 /// <summary>
00850 /// The number of intervals between grid lines.  Note that the number of grid lines will be one
      greater than this.
00851 /// </summary>
00852         public int IntervalCount { get; set; } = 10;
00853
00854 /// <summary>
00855 /// The coordinate system used to transform the points from data space to plot space.
00856 /// </summary>
00857         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00858         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00859
00860 /// <summary>
00861 /// Presentation attributes determining the appearance of the grid.
00862 /// </summary>
00863         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00864
00865 /// <summary>
00866 /// A tag to identify the grid in the plot.
00867 /// </summary>
00868         public string Tag { get; set; }
00869
00870 /// <summary>
00871 /// Create a new <see cref="Grid"/> instance.
00872 /// </summary>
00873 /// <param name="side1Start">The starting point for the first side of the grid.</param>
00874 /// <param name="side1End">The ending point for the first side of the grid.</param>
00875 /// <param name="side2Start">The starting point for the second side of the grid.</param>
00876 /// <param name="side2End">The ending point for the second side of the grid.</param>
00877 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00878         public Grid(IReadOnlyList<double> side1Start, IReadOnlyList<double> side1End,
      IReadOnlyList<double> side2Start, IReadOnlyList<double> side2End, IContinuousCoordinateSystem
      coordinateSystem)
00879         {
00880             this.Side1Start = side1Start;
00881             this.Side1End = side1End;
00882             this.Side2Start = side2Start;
00883             this.Side2End = side2End;
00884             this.CoordinateSystem = coordinateSystem;
00885         }
00886
00887 /// <inheritdoc/>
00888         public void Plot(Graphics target)
00889         {
00890             IReadOnlyList<double> StartPoint1 = Side1Start;
00891             IReadOnlyList<double> EndPoint1 = Side1End;
```

```
00892             IReadOnlyList<double> StartPoint2 = Side2Start;
00893             IReadOnlyList<double> EndPoint2 = Side2End;
00894 
00895             double[] direction1 = new double[StartPoint1.Count];
00896 
00897             for (int i = 0; i < direction1.Length; i++)
00898             {
00899                 direction1[i] = EndPoint1[i] - StartPoint1[i];
00900             }
00901 
00902             double[] direction2 = new double[StartPoint2.Count];
00903 
00904             for (int i = 0; i < direction2.Length; i++)
00905             {
00906                 direction2[i] = EndPoint2[i] - StartPoint2[i];
00907             }
00908 
00909             GraphicsPath gridPath = new GraphicsPath();
00910 
00911             for (int i = 0; i <= IntervalCount; i++)
00912             {
00913                 double[] pt1 = new double[StartPoint1.Count];
00914                 double[] pt2 = new double[StartPoint2.Count];
00915 
00916                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv)
00917                 {
00918                     if (inv.IsDirectionStraight(direction1))
00919                     {
00920                         Point start = inv.ToPlotCoordinates(StartPoint1);
00921                         Point end = inv.ToPlotCoordinates(EndPoint1);
00922 
00923                         pt1 = inv.ToDataCoordinates(new Point(start.X * (1 - (double)i /
      IntervalCount) + end.X * i / IntervalCount, start.Y * (1 - (double)i / IntervalCount) + end.Y * i /
      IntervalCount));
00924                     }
00925                     else
00926                     {
00927                         for (int j = 0; j < pt1.Length; j++)
00928                         {
00929                             pt1[j] = StartPoint1[j] + direction1[j] / IntervalCount * i;
00930                         }
00931                     }
00932 
00933                     if (inv.IsDirectionStraight(direction2))
00934                     {
00935                         Point start = inv.ToPlotCoordinates(StartPoint2);
00936                         Point end = inv.ToPlotCoordinates(EndPoint2);
00937 
00938                         pt2 = inv.ToDataCoordinates(new Point(start.X * (1 - (double)i /
      IntervalCount) + end.X * i / IntervalCount, start.Y * (1 - (double)i / IntervalCount) + end.Y * i /
      IntervalCount));
00939                     }
00940                     else
00941                     {
00942                         for (int j = 0; j < pt1.Length; j++)
00943                         {
00944                             pt2[j] = StartPoint2[j] + direction2[j] / IntervalCount * i;
00945                         }
00946                     }
00947                 }
00948                 else
00949                 {
00950                     for (int j = 0; j < pt1.Length; j++)
00951                     {
00952                         pt1[j] = StartPoint1[j] + direction1[j] / IntervalCount * i;
00953                         pt2[j] = StartPoint2[j] + direction2[j] / IntervalCount * i;
00954                     }
00955                 }
00956 
00957 
00958 
00959 
00960                 Point point1 = CoordinateSystem.ToPlotCoordinates(pt1);
00961                 Point point2 = CoordinateSystem.ToPlotCoordinates(pt2);
00962 
00963 
00964                 gridPath.MoveTo(point1).LineTo(point2);
00965             }
00966 
00967             target.StrokePath(gridPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  Tag);
00968         }
00969     }
00970 }
```

8.22 Bars.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025 /// <summary>
00026 /// A plot element that draws bars.
00027 /// </summary>
00028 /// <typeparam name="T">The type of data elements.</typeparam>
00029     public class Bars<T> :  IPlotElement
00030     {
00031         private double margin = 0;
00032
00033 /// <summary>
00034 /// The data points corresponding to the tips of the bars.
00035 /// </summary>
00036         public SortedSet<T> Data { get; set; }
00037
00038 /// <summary>
00039 /// A function that returns the bottom for each bar.  This function should accept
00040 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00041 /// object, representing the bottom of the bar in data space.
00042 /// </summary>
00043         public Func<T, T> GetBaseline { get; set; }
00044
00045 /// <summary>
00046 /// The margin between consecutive bars.  This should range between 0 and 1.
00047 /// </summary>
00048         public double Margin
00049         {
00050             get => margin;
00051             set
00052             {
00053                 if (value >= 0 && value <= 1)
00054                 {
00055                     margin = value;
00056                 }
00057                 else
00058                 {
00059                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00060                 }
00061             }
00062         }
00063
00064 /// <summary>
00065 /// The coordinate system used to transform the points from data space to plot space.
00066 /// </summary>
00067         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00068         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00069
00070 /// <summary>
00071 /// Presentation attributes for the bars.
00072 /// </summary>
00073         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00074
00075 /// <summary>
00076 /// A tag to identify the bars in the plot.
00077 /// </summary>
00078         public string Tag { get; set; }
00079
00080 /// <summary>
00081 /// Create a new <see cref="Bars{T}"/> instance.
00082 /// </summary>
00083 /// <param name="data">The data points corresponding to the tips of the bars.</param>
```

```
00084 /// <param name="sorting">A comparer used to sort the bars.</param>
00085 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00086 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00087 /// object, representing the bottom of the bar in data space.</param>
00088 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00089         public Bars(IEnumerable<T> data, IComparer<T> sorting, Func<T, T> getBaseline,
      ICoordinateSystem<T> coordinateSystem)
00090         {
00091             this.Data = new SortedSet<T>(data, sorting);
00092             this.CoordinateSystem = coordinateSystem;
00093             this.GetBaseline = getBaseline;
00094         }
00095
00096 /// <summary>
00097 /// Create a new <see cref="Bars{T}"/> instance.
00098 /// </summary>
00099 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00100 /// <param name="sorting">A comparer used to sort the bars.</param>
00101 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00102 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00103 /// object, representing the bottom of the bar in data space.</param>
00104 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00105         public Bars(IEnumerable<T> data, Comparison<T> sorting, Func<T, T> getBaseline,
      ICoordinateSystem<T> coordinateSystem) :  this(data, Comparer<T>.Create(sorting), getBaseline,
      coordinateSystem) { }
00106
00107 /// <summary>
00108 /// Create a new <see cref="Bars{T}"/> instance.
00109 /// </summary>
00110 /// <param name="data">The data points corresponding to the tips of the bars.  These should already be
      sorted.</param>
00111 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00112 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00113 /// object, representing the bottom of the bar in data space.</param>
00114 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00115         public Bars(IReadOnlyList<T> data, Func<T, T> getBaseline, ICoordinateSystem<T>
      coordinateSystem)
00116         {
00117             Dictionary<T, int> indices = new Dictionary<T, int>();
00118
00119             for (int i = 0; i < data.Count; i++)
00120             {
00121                 indices[data[i]] = i;
00122             }
00123
00124             this.Data = new SortedSet<T>(data, Comparer<T>.Create((x, y) => Math.Sign(indices[x] -
      indices[y])));
00125             this.CoordinateSystem = coordinateSystem;
00126             this.GetBaseline = getBaseline;
00127         }
00128
00129         private static bool AnyNan(params Point[] points)
00130         {
00131             for (int i = 0; i < points.Length; i++)
00132             {
00133                 if (double.IsNaN(points[i].X) || double.IsNaN(points[i].Y))
00134                 {
00135                     return true;
00136                 }
00137             }
00138
00139             return false;
00140         }
00141
00142 /// <inheritdoc/>
00143         public void Plot(Graphics target)
00144         {
00145             List<(Point, Point)> bars = new List<(Point, Point)>();
00146
00147             foreach (T data in Data)
00148             {
00149                 Point pt = CoordinateSystem.ToPlotCoordinates(data);
00150                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00151                 bars.Add((baseline, pt));
00152             }
00153
00154             for (int i = 0; i < bars.Count; i++)
00155             {
00156                 GraphicsPath pth = new GraphicsPath();
00157
00158                 if (i > 0 && i < bars.Count - 1)
```

```
00159                 {
00160                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00161
00162                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00163                     perpDir = perpDir.Normalize();
00164
00165                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00166                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00167
00168                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00169
00170                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00171                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00172
00173                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00174                     {
00175
      pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00176                     }
00177                 }
00178                 else if (i == 0)
00179                 {
00180                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00181                     perpDir = perpDir.Normalize();
00182
00183                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00184
00185                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00186                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00187
00188
00189                     Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00190                     double t = -t2;
00191                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00192
00193                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00194                     {
00195
      pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00196                     }
00197                 }
00198                 else if (i == bars.Count - 1)
00199                 {
00200                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00201
00202                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00203                     perpDir = perpDir.Normalize();
00204
00205                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00206                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00207
00208                     Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
      bars[i].Item1.Y - prevBaselineMid.Y);
00209
00210                     double t2 = -t;
00211                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00212
00213                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00214                     {
00215
      pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00216                     }
00217                 }
00218
00219                 string tag = Tag;
```

```
00220                 string strokeTag = Tag;
00221
00222                 if (!string.IsNullOrEmpty(Tag))
00223                 {
00224                     tag = tag + "@" + i.ToString();
00225                     strokeTag = tag + "@" + i.ToString() + "_stroke";
00226                 }
00227
00228                 if (PresentationAttributes.Fill != null)
00229                 {
00230                     target.FillPath(pth, PresentationAttributes.Fill, tag);
00231                 }
00232
00233                 if (PresentationAttributes.Stroke != null)
00234                 {
00235                     target.StrokePath(pth, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00236                 }
00237             }
00238         }
00239     }
00240
00241 /// <summary>
00242 /// A plot element that draws bars for categorical data.
00243 /// </summary>
00244 /// <typeparam name="T">The type of the data categories.</typeparam>
00245     public class CategoricalBars<T> :  Bars<(T, double)>
00246     {
00247 /// <summary>
00248 /// Create a new <see cref="CategoricalBars{T}"/> instance.
00249 /// </summary>
00250 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00251 /// <param name="sorting">A comparer used to sort the bars.</param>
00252 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00253 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00254 /// object, representing the bottom of the bar in data space.</param>
00255 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00256         public CategoricalBars(IEnumerable<(T, double)> data, IComparer<(T, double)> sorting, Func<(T,
      double), (T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem) :  base(data,
      sorting, getBaseline, coordinateSystem) { }
00257
00258 /// <summary>
00259 /// Create a new <see cref="CategoricalBars{T}"/> instance.
00260 /// </summary>
00261 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00262 /// <param name="sorting">A comparer used to sort the bars.</param>
00263 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00264 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00265 /// object, representing the bottom of the bar in data space.</param>
00266 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00267         public CategoricalBars(IEnumerable<(T, double)> data, Comparison<(T, double)> sorting,
      Func<(T, double), (T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem) :
      base(data, sorting, getBaseline, coordinateSystem) { }
00268
00269 /// <summary>
00270 /// Create a new <see cref="Bars{T}"/> instance.  The baseline for each <c>(<typeparamref name="T"/>
      x, <see langword="double"/> y)</c> is determined automatically as <c>(x, 0)</c>.
00271 /// </summary>
00272 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00273 /// <param name="sorting">A comparer used to sort the bars.</param>
00274 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00275         public CategoricalBars(IEnumerable<(T, double)> data, Comparison<(T, double)> sorting,
      ICoordinateSystem<(T, double)> coordinateSystem) :  base(data, sorting, x => (x.Item1, 0),
      coordinateSystem) { }
00276
00277 /// <summary>
00278 /// Create a new <see cref="Bars{T}"/> instance.  The baseline for each <c>(<typeparamref name="T"/>
      x, <see langword="double"/> y)</c> is determined automatically as <c>(x, 0)</c>.
00279 /// </summary>
00280 /// <param name="data">The data points corresponding to the tips of the bars.  These should already be
      sorted.</param>
00281 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00282         public CategoricalBars(IReadOnlyList<(T, double)> data, ICoordinateSystem<(T, double)>
      coordinateSystem) :  base(data, x => (x.Item1, 0), coordinateSystem) { }
00283     }
00284
00285 /// <summary>
00286 /// A plot element that draws bars for numerical data.
00287 /// </summary>
00288     public class Bars :  Bars<IReadOnlyList<double»
```

```
00289     {
00290 /// <summary>
00291 /// Create a new <see cref="Bars"/> instance.
00292 /// </summary>
00293 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00294 /// <param name="sorting">A comparer used to sort the bars.</param>
00295 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00296 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00297 /// object of the same type, representing the bottom of the bar in data space.</param>
00298 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00299         public Bars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double» sorting,
      Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem) :  base(data, sorting, getBaseline, coordinateSystem) { }
00300
00301 /// <summary>
00302 /// Create a new <see cref="Bars"/> instance.
00303 /// </summary>
00304 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00305 /// <param name="sorting">A comparer used to sort the bars.</param>
00306 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00307 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00308 /// object of the same type, representing the bottom of the bar in data space.</param>
00309 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00310         public Bars(IEnumerable<IReadOnlyList<double» data, Comparison<IReadOnlyList<double» sorting,
      Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem) :  base(data, sorting, getBaseline, coordinateSystem) { }
00311
00312 /// <summary>
00313 /// Create a new <see cref="Bars"/> instance.
00314 /// </summary>
00315 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00316 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00317 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
      above the X axis
00318 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00319         public Bars(IEnumerable<IReadOnlyList<double» data, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem, bool vertical = true) :  base(data, vertical ?
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[0] - y[0])) :
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
      return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr =
      x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { }
00320     }
00321
00322 /// <summary>
00323 /// A plot element that draws stacked bars.
00324 /// </summary>
00325     public class StackedBars :  IPlotElement
00326     {
00327         private double margin = 0;
00328
00329 /// <summary>
00330 /// If this is <see langword="true"/>, the bars rise vertically above the X axis
00331 /// Otherwise, the bars grow horizontally from the Y axis.
00332 /// </summary>
00333         public bool Vertical { get; set; } = true;
00334
00335 /// <summary>
00336 /// The data points corresponding to the tips of the bars.  For each bar stack, the data
00337 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00338 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00339 /// length of each segment in the bar stack.
00340 /// </summary>
00341         public SortedSet<IReadOnlyList<double» Data { get; set; }
00342
00343 /// <summary>
00344 /// A function that returns the bottom for each bar.  This function should accept
00345 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00346 /// object of the same type, representing the bottom of the bar in data space.
00347 /// </summary>
00348         public Func<IReadOnlyList<double>, IReadOnlyList<double» GetBaseline { get; set; }
00349
00350 /// <summary>
00351 /// The margin between consecutive bars.  This should range between 0 and 1.
00352 /// </summary>
00353         public double Margin
00354         {
00355             get => margin;
```

```
00356             set
00357             {
00358                 if (value >= 0 && value <= 1)
00359                 {
00360                     margin = value;
00361                 }
00362                 else
00363                 {
00364                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00365                 }
00366             }
00367         }
00368
00369 /// <summary>
00370 /// The coordinate system used to transform the points from data space to plot space.
00371 /// </summary>
00372         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00373         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00374
00375 /// <summary>
00376 /// Presentation attributes for the bars.  An element from this collection is used for each segment in
00377 /// the stack; if there are more segments than elements in this collection, the presentation
      attributes
00378 /// are wrapped.
00379 /// </summary>
00380         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00381
00382 /// <summary>
00383 /// A tag to identify the stacked bars in the plot.
00384 /// </summary>
00385         public string Tag { get; set; }
00386
00387 /// <summary>
00388 /// Create a new <see cref="StackedBars"/> instance.
00389 /// </summary>
00390 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
00391 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00392 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00393 /// length of each segment in the bar stack.</param>
00394 /// <param name="sorting">A comparer used to sort the bars.</param>
00395 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00396 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00397 /// object of the same type, representing the bottom of the bar in data space.</param>
00398 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00399         public StackedBars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00400         {
00401             this.Data = new SortedSet<IReadOnlyList<double»(data, sorting);
00402             this.CoordinateSystem = coordinateSystem;
00403             this.GetBaseline = getBaseline;
00404         }
00405
00406 /// <summary>
00407 /// Create a new <see cref="StackedBars"/> instance.
00408 /// </summary>
00409 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
00410 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00411 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00412 /// length of each segment in the bar stack.</param>
00413 /// <param name="sorting">A comparer used to sort the bars.</param>
00414 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00415 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00416 /// object of the same type, representing the bottom of the bar in data space.</param>
00417 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00418         public StackedBars(IEnumerable<IReadOnlyList<double» data, Comparison<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem) :  this(data,
      Comparer<IReadOnlyList<double».Create(sorting), getBaseline, coordinateSystem) { }
00419
00420 /// <summary>
00421 /// Create a new <see cref="StackedBars"/> instance.
00422 /// </summary>
00423 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
```

```
00424 /// point contains an element determining the position of the bar on the X axis (if <paramref
      name="vertical"/>
00425 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00426 /// length of each segment in the bar stack.</param>
00427 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00428 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
      above the X axis
00429 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00430         public StackedBars(IEnumerable<IReadOnlyList<double>> data,
      ICoordinateSystem<IReadOnlyList<double>> coordinateSystem, bool vertical = true) :  this(data, vertical
      ?  Comparer<IReadOnlyList<double>.Create((x, y) => Math.Sign(x[0] - y[0])) :
      Comparer<IReadOnlyList<double>.Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double>>(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
      return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double>>(x => { double[] tbr =
      x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { Vertical = vertical; }
00431
00432 /// <inheritdoc/>
00433         public void Plot(Graphics target)
00434         {
00435             List<(Point, Point)> bars = new List<(Point, Point)>();
00436             List<double[]> ratios = new List<double[]>();
00437
00438             foreach (IReadOnlyList<double> data in Data)
00439             {
00440                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00441
00442                 double total = 0;
00443
00444                 double[] currRatios = new double[data.Count - 1];
00445
00446                 for (int i = 0; i < data.Count - 1; i++)
00447                 {
00448                     if (i == 0)
00449                     {
00450                         total += Vertical ?  data[1] :  data[0];
00451                     }
00452                     else
00453                     {
00454                         total += data[i + 1];
00455                     }
00456
00457                     Point currPt = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
      data[0], total } :  new double[] { total, data[1] });
00458
00459                     currRatios[i] = new Point(currPt.X - baseline.X, currPt.Y - baseline.Y).Modulus();
00460                 }
00461
00462                 for (int i = 0; i < currRatios.Length; i++)
00463                 {
00464                     currRatios[i] /= currRatios[currRatios.Length - 1];
00465                 }
00466
00467                 Point pt = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] { data[0],
      total } :  new double[] { total, data[1] });
00468
00469                 bars.Add((baseline, pt));
00470                 ratios.Add(currRatios);
00471             }
00472
00473             for (int i = 0; i < bars.Count; i++)
00474             {
00475                 List<GraphicsPath> paths = new List<GraphicsPath>();
00476
00477                 if (i > 0 && i < bars.Count - 1)
00478                 {
00479                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00480
00481                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00482                     perpDir = perpDir.Normalize();
00483
00484                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00485                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00486
00487                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00488
00489                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00490                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
```

```
00491
00492                     Point lastBaseLeft = prevBaselineMid;
00493                     Point lastBaseRight = nextBaselineMid;
00494
00495                     for (int j = 0; j < ratios[i].Length; j++)
00496                     {
00497                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00498                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00499
00500                         paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00501
00502                         lastBaseLeft = pTop;
00503                         lastBaseRight = nTop;
00504                     }
00505                 }
00506                 else if (i == 0)
00507                 {
00508                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00509                     perpDir = perpDir.Normalize();
00510
00511                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00512
00513                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00514                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00515
00516
00517                     Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00518                     double t = -t2;
00519                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00520
00521                     Point lastBaseLeft = prevBaselineMid;
00522                     Point lastBaseRight = nextBaselineMid;
00523
00524                     for (int j = 0; j < ratios[i].Length; j++)
00525                     {
00526                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00527                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00528
00529                         paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00530
00531                         lastBaseLeft = pTop;
00532                         lastBaseRight = nTop;
00533                     }
00534                 }
00535                 else if (i == bars.Count - 1)
00536                 {
00537                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00538
00539                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00540                     perpDir = perpDir.Normalize();
00541
00542                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00543                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00544
00545                     Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
      bars[i].Item1.Y - prevBaselineMid.Y);
00546
00547                     double t2 = -t;
00548                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00549
00550                     Point lastBaseLeft = prevBaselineMid;
00551                     Point lastBaseRight = nextBaselineMid;
00552
00553                     for (int j = 0; j < ratios[i].Length; j++)
00554                     {
00555                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00556                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
```

```
    ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00557
00558                       paths.Add(new
    GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00559
00560                       lastBaseLeft = pTop;
00561                       lastBaseRight = nTop;
00562                   }
00563               }
00564
00565               for (int j = 0; j < paths.Count; j++)
00566               {
00567                   string tag = Tag;
00568                   string strokeTag = Tag;
00569
00570                   if (!string.IsNullOrEmpty(Tag))
00571                   {
00572                       tag = tag + "@" + i.ToString() + "/" + j.ToString();
00573                       strokeTag = tag + "@" + i.ToString() + "/" + j.ToString() + "_stroke";
00574                   }
00575
00576                   int colourIndex = j % PresentationAttributes.Count;
00577
00578                   if (PresentationAttributes[colourIndex].Fill != null)
00579                   {
00580                       target.FillPath(paths[j], PresentationAttributes[colourIndex].Fill, tag);
00581                   }
00582
00583                   if (PresentationAttributes[colourIndex].Stroke != null)
00584                   {
00585                       target.StrokePath(paths[j], PresentationAttributes[colourIndex].Stroke,
    PresentationAttributes[colourIndex].LineWidth, PresentationAttributes[colourIndex].LineCap,
    PresentationAttributes[colourIndex].LineJoin, PresentationAttributes[colourIndex].LineDash,
    strokeTag);
00586                   }
00587               }
00588           }
00589       }
00590   }
00591
00592 /// <summary>
00593 /// A plot element that draws clusters of bars.
00594 /// </summary>
00595   public class ClusteredBars :  IPlotElement
00596   {
00597       private double interClusterMargin = 0;
00598       private double intraClusterMargin = 0;
00599
00600 /// <summary>
00601 /// If this is <see langword="true"/>, the bars rise vertically above the X axis
00602 /// Otherwise, the bars grow horizontally from the Y axis.
00603 /// </summary>
00604       public bool Vertical { get; set; } = true;
00605
00606
00607 /// <summary>
00608 /// The data points corresponding to the tips of the bars.  For each bar cluster, the data
00609 /// point contains an element determining the position of the cluster on the X axis (if <see
    cref="Vertical"/>
00610 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00611 /// length of each bar in the cluster.
00612 /// </summary>
00613       public SortedSet<IReadOnlyList<double>> Data { get; set; }
00614
00615 /// <summary>
00616 /// A function that returns the bottom for each bar cluster.  This function should accept
00617 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
    another
00618 /// object of the same type, representing the bottom of the cluster in data space.
00619 /// </summary>
00620       public Func<IReadOnlyList<double>, IReadOnlyList<double>> GetBaseline { get; set; }
00621
00622 /// <summary>
00623 /// The margin between consecutive bar clusters.
00624 /// </summary>
00625       public double InterClusterMargin
00626       {
00627           get => interClusterMargin;
00628           set
00629           {
00630               if (value >= 0 && value <= 1)
00631               {
00632                   interClusterMargin = value;
00633               }
00634               else
00635               {
00636                   throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
```

```
margin must be within 0 and 1 (inclusive)!");
00637                 }
00638             }
00639         }
00640 
00641 /// <summary>
00642 /// The margin between consecutive bars within a single cluster.
00643 /// </summary>
00644         public double IntraClusterMargin
00645         {
00646             get => intraClusterMargin;
00647             set
00648             {
00649                 if (value >= 0 && value <= 1)
00650                 {
00651                     intraClusterMargin = value;
00652                 }
00653                 else
00654                 {
00655                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
margin must be within 0 and 1 (inclusive)!");
00656                 }
00657             }
00658         }
00659 
00660 /// <summary>
00661 /// The coordinate system used to transform the points from data space to plot space.
00662 /// </summary>
00663         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00664         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00665 
00666 /// <summary>
00667 /// Presentation attributes for the bars.  An element from this collection is used for each bar in
00668 /// the cluster; if there are more bars than elements in this collection, the presentation attributes
00669 /// are wrapped.
00670 /// </summary>
00671         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00672 
00673 /// <summary>
00674 /// A tag to identify the clustered bars in the plot.
00675 /// </summary>
00676         public string Tag { get; set; }
00677 
00678 
00679 /// <summary>
00680 /// Create a new <see cref="ClusteredBars"/> instance.
00681 /// </summary>
00682 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
the data
00683 /// point contains an element determining the position of the cluster on the X axis (if <see
cref="Vertical"/>
00684 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00685 /// length of each bar in the cluster.</param>
00686 /// <param name="sorting">A comparer used to sort the bar clusters.</param>
00687 /// <param name="getBaseline">A function that returns the bottom for each bar cluster.  This function
should accept
00688 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
another
00689 /// object of the same type, representing the bottom of the cluster in data space.</param>
00690 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00691         public ClusteredBars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double»
sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00692         {
00693             this.Data = new SortedSet<IReadOnlyList<double»(data, sorting);
00694             this.CoordinateSystem = coordinateSystem;
00695             this.GetBaseline = getBaseline;
00696         }
00697 
00698 /// <summary>
00699 /// Create a new <see cref="ClusteredBars"/> instance.
00700 /// </summary>
00701 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
the data
00702 /// point contains an element determining the position of the cluster on the X axis (if <see
cref="Vertical"/>
00703 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00704 /// length of each bar in the cluster.</param>
00705 /// <param name="sorting">A comparer used to sort the bar clusters.</param>
00706 /// <param name="getBaseline">A function that returns the bottom for each bar cluster.  This function
should accept
00707 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
another
00708 /// object of the same type, representing the bottom of the cluster in data space.</param>
00709 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
```

```
to plot space.</param>
00710         public ClusteredBars(IEnumerable<IReadOnlyList<double> data, Comparison<IReadOnlyList<double>
sorting, Func<IReadOnlyList<double>, IReadOnlyList<double> getBaseline,
ICoordinateSystem<IReadOnlyList<double> coordinateSystem) :  this(data,
Comparer<IReadOnlyList<double>.Create(sorting), getBaseline, coordinateSystem) { }
00711
00712 /// <summary>
00713 /// Create a new <see cref="ClusteredBars"/> instance.
00714 /// </summary>
00715 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
the data
00716 /// point contains an element determining the position of the cluster on the X axis (if <see
cref="Vertical"/>
00717 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00718 /// length of each bar in the cluster.</param>
00719 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00720 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
above the X axis
00721 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00722         public ClusteredBars(IEnumerable<IReadOnlyList<double> data,
ICoordinateSystem<IReadOnlyList<double> coordinateSystem, bool vertical = true) :  this(data, vertical
?  Comparer<IReadOnlyList<double>.Create((x, y) => Math.Sign(x[0] - y[0])) :
Comparer<IReadOnlyList<double>.Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
Func<IReadOnlyList<double>, IReadOnlyList<double>(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double>(x => { double[] tbr =
x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { Vertical = vertical; }
00723
00724 /// <inheritdoc/>
00725         public void Plot(Graphics target)
00726         {
00727             List<(Point, Point[])> bars = new List<(Point, Point[])>();
00728
00729             foreach (IReadOnlyList<double> data in Data)
00730             {
00731                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00732
00733                 Point[] points = new Point[data.Count - 1];
00734
00735                 for (int i = 0; i < data.Count - 1; i++)
00736                 {
00737                     if (i == 0)
00738                     {
00739                         points[i] = CoordinateSystem.ToPlotCoordinates(data);
00740                     }
00741                     else
00742                     {
00743                         points[i] = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
data[0], data[i + 1] } :  new double[] { data[i + 1], data[1] });
00744                     }
00745                 }
00746
00747                 bars.Add((baseline, points));
00748             }
00749
00750             for (int i = 0; i < bars.Count; i++)
00751             {
00752                 List<GraphicsPath> paths = new List<GraphicsPath>();
00753
00754                 if (i > 0 && i < bars.Count - 1)
00755                 {
00756                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
0.5) + bars[i - 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
InterClusterMargin * 0.5) + bars[i - 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00757                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
0.5) + bars[i + 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
InterClusterMargin * 0.5) + bars[i + 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00758
00759                     for (int j = 0; j < bars[i].Item2.Length; j++)
00760                     {
00761                         Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
-(bars[i].Item2[j].X - bars[i].Item1.X));
00762                         perpDir = perpDir.Normalize();
00763
00764                         double t = (bars[i].Item2[j].X - prevBaselineMid.X) * perpDir.X +
(bars[i].Item2[j].Y - prevBaselineMid.Y) * perpDir.Y;
00765                         Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
bars[i].Item2[j].Y - perpDir.Y * t);
00766
00767                         double t2 = (bars[i].Item2[j].X - nextBaselineMid.X) * perpDir.X +
(bars[i].Item2[j].Y - nextBaselineMid.Y) * perpDir.Y;
00768                         Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
bars[i].Item2[j].Y - perpDir.Y * t2);
00769
00770                         double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00771                         double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
bars[i].Item2.Length;
```

```
00772
00773                         paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
      (nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
      prevBaselineMid.Y) * start))
00774                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
      (nextTop.Y - prevTop.Y) * start))
00775                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
      (nextTop.Y - prevTop.Y) * end))
00776                             .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
      prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
      end)).Close());
00777                     }
00778                 }
00779                 else if (i == 0)
00780                 {
00781                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
      0.5) + bars[i + 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
      InterClusterMargin * 0.5) + bars[i + 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00782
00783                     for (int j = 0; j < bars[i].Item2.Length; j++)
00784                     {
00785                         Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
      -(bars[i].Item2[j].X - bars[i].Item1.X));
00786                         perpDir = perpDir.Normalize();
00787
00788                         double t2 = (bars[i].Item2[j].X - nextBaselineMid.X) * perpDir.X +
      (bars[i].Item2[j].Y - nextBaselineMid.Y) * perpDir.Y;
00789                         Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
      bars[i].Item2[j].Y - perpDir.Y * t2);
00790
00791                         Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00792                         double t = -t2;
00793                         Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
      bars[i].Item2[j].Y - perpDir.Y * t);
00794
00795                         double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00796                         double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
      bars[i].Item2.Length;
00797
00798                         paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
      (nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
      prevBaselineMid.Y) * start))
00799                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
      (nextTop.Y - prevTop.Y) * start))
00800                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
      (nextTop.Y - prevTop.Y) * end))
00801                             .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
      prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
      end)).Close());
00802                     }
00803                 }
00804                 else if (i == bars.Count - 1)
00805                 {
00806                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
      0.5) + bars[i - 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
      InterClusterMargin * 0.5) + bars[i - 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00807
00808                     for (int j = 0; j < bars[i].Item2.Length; j++)
00809                     {
00810                         Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
      -(bars[i].Item2[j].X - bars[i].Item1.X));
00811                         perpDir = perpDir.Normalize();
00812
00813                         double t = (bars[i].Item2[j].X - prevBaselineMid.X) * perpDir.X +
      (bars[i].Item2[j].Y - prevBaselineMid.Y) * perpDir.Y;
00814                         Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
      bars[i].Item2[j].Y - perpDir.Y * t);
00815
00816                         Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
      bars[i].Item1.Y - prevBaselineMid.Y);
00817
00818                         double t2 = -t;
00819                         Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
      bars[i].Item2[j].Y - perpDir.Y * t2);
00820
00821                         double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00822                         double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
      bars[i].Item2.Length;
00823
00824                         paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
      (nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
      prevBaselineMid.Y) * start))
00825                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
      (nextTop.Y - prevTop.Y) * start))
00826                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
      (nextTop.Y - prevTop.Y) * end))
```

```
00827                             .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
      prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
      end)).Close());
00828                     }
00829                 }
00830
00831                 for (int j = 0; j < paths.Count; j++)
00832                 {
00833                     string tag = Tag;
00834                     string strokeTag = Tag;
00835
00836                     if (!string.IsNullOrEmpty(Tag))
00837                     {
00838                         tag = tag + "@" + i.ToString() + "/" + j.ToString();
00839                         strokeTag = tag + "@" + i.ToString() + "/" + j.ToString() + "_stroke";
00840                     }
00841
00842                     int colourIndex = j % PresentationAttributes.Count;
00843
00844                     if (PresentationAttributes[colourIndex].Fill != null)
00845                     {
00846                         target.FillPath(paths[j], PresentationAttributes[colourIndex].Fill, tag);
00847                     }
00848
00849                     if (PresentationAttributes[colourIndex].Stroke != null)
00850                     {
00851                         target.StrokePath(paths[j], PresentationAttributes[colourIndex].Stroke,
      PresentationAttributes[colourIndex].LineWidth, PresentationAttributes[colourIndex].LineCap,
      PresentationAttributes[colourIndex].LineJoin, PresentationAttributes[colourIndex].LineDash,
      strokeTag);
00852                     }
00853                 }
00854             }
00855         }
00856     }
00857 }
```

8.23 BoxPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// A plot element that draws a box plot.
00025 /// </summary>
00026     public class BoxPlot :  IPlotElement
00027     {
00028 /// <summary>
00029 /// The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
00030 /// </summary>
00031         public IReadOnlyList<double> Position { get; set; }
00032
00033 /// <summary>
00034 /// The direction along which the box plot is drawn, in data space coordinates.
00035 /// </summary>
00036         public IReadOnlyList<double> Direction { get; set; }
00037
00038 /// <summary>
00039 /// The distance between the centre of the box plot and the first whisker, expressed as a multiple
00040 /// of <see cref="Direction"/>.
00041 /// </summary>
00042         public double Whisker1 { get; set; }
00043
00044 /// <summary>
```

```
00045 /// The distance between the centre of the box plot and the first side of the box, expressed as a
      multiple
00046 /// of <see cref="Direction"/>.
00047 /// </summary>
00048         public double Box1 { get; set; }
00049
00050 /// <summary>
00051 /// The distance between the centre of the box plot and the second side of the box, expressed as a
      multiple
00052 /// of <see cref="Direction"/>.
00053 /// </summary>
00054         public double Box2 { get; set; }
00055
00056 /// <summary>
00057 /// The distance between the centre of the box plot and the second whisker, expressed as a multiple
00058 /// of <see cref="Direction"/>.
00059 /// </summary>
00060         public double Whisker2 { get; set; }
00061
00062 /// <summary>
00063 /// The width of the box plot, in data space coordinates.
00064 /// </summary>
00065         public double Width { get; set; } = 10;
00066
00067 /// <summary>
00068 /// The width of the whiskers, expressed as a multiple of the <see cref="Width"/> of the box.
00069 /// </summary>
00070         public double WhiskerWidth { get; set; } = 0.5;
00071
00072 /// <summary>
00073 /// The width of the notch, expressed as a multiple of the <see cref="Width"/> of the box.
00074 /// </summary>
00075         public double NotchWidth { get; set; } = 0.5;
00076
00077 /// <summary>
00078 /// The distance between the centre of the box plot and the end of the notch, expressed as a multiple
00079 /// of <see cref="Direction"/>.
00080 /// </summary>
00081         public double NotchSize { get; set; } = 0;
00082
00083 /// <summary>
00084 /// Presentation attributes for the box.
00085 /// </summary>
00086         public PlotElementPresentationAttributes BoxPresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = Colours.White };
00087
00088 /// <summary>
00089 /// Presentation attributes for the whiskers.
00090 /// </summary>
00091         public PlotElementPresentationAttributes WhiskersPresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = null };
00092
00093 /// <summary>
00094 /// Presentation attributes for the symbol drawn at the centre of the box plot.
00095 /// </summary>
00096         public PlotElementPresentationAttributes CentreSymbolPresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes() { Fill = null, Stroke = null };
00097
00098 /// <summary>
00099 /// Symbol drawn at the centre of the box plot.
00100 /// </summary>
00101         public IDataPointElement CentreSymbol { get; set; } = new PathDataPointElement();
00102
00103 /// <summary>
00104 /// The coordinate system used to transform the points from data space to plot space.
00105 /// </summary>
00106         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00107
00108 /// <summary>
00109 /// A tag to identify the box in the plot.
00110 /// </summary>
00111         public string Tag { get; set; }
00112
00113         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00114
00115 /// <summary>
00116 /// Create a new <see cref="BoxPlot"/> instance.
00117 /// </summary>
00118 /// <param name="position">The position of the centre (e.g., median or mean) of the box plot, in data
      space coordinates.</param>
00119 /// <param name="direction">The direction along which the box plot is drawn, in data space
      coordinates.</param>
00120 /// <param name="whisker1">The distance between the centre of the box plot and the first whisker,
      expressed as a multiple
00121 /// of <paramref name="direction"/>.</param>
00122 /// <param name="box1">The distance between the centre of the box plot and the first side of the box,
      expressed as a multiple
```

```
00123 /// of <paramref name="direction"/>.</param>
00124 /// <param name="box2">The distance between the centre of the box plot and the second side of the box,
      expressed as a multiple
00125 /// of <paramref name="direction"/>.</param>
00126 /// <param name="whisker2">The distance between the centre of the box plot and the second whisker,
      expressed as a multiple
00127 /// of <paramref name="direction"/>.</param>
00128 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00129         public BoxPlot(IReadOnlyList<double> position, IReadOnlyList<double> direction, double
      whisker1, double box1, double box2, double whisker2, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem)
00130         {
00131             this.Position = position;
00132             this.Direction = direction;
00133             this.Whisker1 = whisker1;
00134             this.Box1 = box1;
00135             this.Box2 = box2;
00136             this.Whisker2 = whisker2;
00137             this.CoordinateSystem = coordinateSystem;
00138         }
00139
00140 /// <inheritdoc/>
00141         public void Plot(Graphics target)
00142         {
00143             double[] perpDirection = new double[] { -Direction[1], Direction[0] };
00144
00145             Point whisker1 = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1, this.Position[1] + this.Direction[1] * this.Whisker1 });
00146             Point whisker2 = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2, this.Position[1] + this.Direction[1] * this.Whisker2 });
00147
00148             Point whisker1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1 + perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker1 + perpDirection[1] * this.Width *
      this.WhiskerWidth });
00149             Point box1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box1 + perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box1 + perpDirection[1] * this.Width });
00150             Point box2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box2 + perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box2 + perpDirection[1] * this.Width });
00151             Point whisker2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2 + perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker2 + perpDirection[1] * this.Width *
      this.WhiskerWidth });
00152
00153             Point whisker1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1 - perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker1 - perpDirection[1] * this.Width *
      this.WhiskerWidth });
00154             Point box1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box1 - perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box1 - perpDirection[1] * this.Width });
00155             Point box2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box2 - perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box2 - perpDirection[1] * this.Width });
00156             Point whisker2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2 - perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker2 - perpDirection[1] * this.Width *
      this.WhiskerWidth });
00157
00158             Point medianLeft = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      perpDirection[0] * this.Width, this.Position[1] + perpDirection[1] * this.Width });
00159             Point medianRight = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] -
      perpDirection[0] * this.Width, this.Position[1] - perpDirection[1] * this.Width });
00160
00161             string whiskerTag = Tag;
00162             string boxFillTag = Tag;
00163             string boxStrokeTag = Tag;
00164             string medianTag = Tag;
00165             string medianSymbolTag = Tag;
00166
00167             if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00168             {
00169                 whiskerTag += "@whiskers";
00170                 boxStrokeTag += "@stroke";
00171                 medianTag += "@median";
00172                 medianSymbolTag += "@medianSymbol";
00173             }
00174
00175             if (NotchSize != 0 && NotchWidth != 1)
00176             {
00177                 medianLeft = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      perpDirection[0] * this.Width * this.NotchWidth, this.Position[1] + perpDirection[1] * this.Width *
      this.NotchWidth });
00178                 medianRight = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] -
```

```
      perpDirection[0] * this.Width * this.NotchWidth, this.Position[1] - perpDirection[1] * this.Width *
      this.NotchWidth });
00179
00180                 Point notch2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
      + this.Direction[0] * this.NotchSize + perpDirection[0] * this.Width, this.Position[1] +
      this.Direction[1] * this.NotchSize + perpDirection[1] * this.Width });
00181                 Point notch1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
      - this.Direction[0] * this.NotchSize + perpDirection[0] * this.Width, this.Position[1] -
      this.Direction[1] * this.NotchSize + perpDirection[1] * this.Width });
00182
00183                 Point notch2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
      + this.Direction[0] * this.NotchSize - perpDirection[0] * this.Width, this.Position[1] +
      this.Direction[1] * this.NotchSize - perpDirection[1] * this.Width });
00184                 Point notch1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
      - this.Direction[0] * this.NotchSize - perpDirection[0] * this.Width, this.Position[1] -
      this.Direction[1] * this.NotchSize - perpDirection[1] * this.Width });
00185
00186
00187                 GraphicsPath whiskers = new GraphicsPath();
00188
00189                 whiskers.MoveTo(whisker1Left).LineTo(whisker1Right);
00190                 whiskers.MoveTo(whisker1).LineTo(whisker2);
00191                 whiskers.MoveTo(whisker2Left).LineTo(whisker2Right);
00192
00193                 GraphicsPath box = new
      GraphicsPath().MoveTo(box1Left).LineTo(box1Right).LineTo(notch1Right).LineTo(medianRight).LineTo(notch2Right).LineTo(box
00194                 GraphicsPath median = new GraphicsPath().MoveTo(medianLeft).LineTo(medianRight);
00195
00196                 if (WhiskersPresentationAttributes.Stroke != null)
00197                 {
00198                     target.StrokePath(whiskers, WhiskersPresentationAttributes.Stroke,
      WhiskersPresentationAttributes.LineWidth, WhiskersPresentationAttributes.LineCap,
      WhiskersPresentationAttributes.LineJoin, WhiskersPresentationAttributes.LineDash, whiskerTag);
00199                 }
00200
00201                 if (BoxPresentationAttributes.Fill != null)
00202                 {
00203                     target.FillPath(box, BoxPresentationAttributes.Fill, boxFillTag);
00204                 }
00205
00206                 if (BoxPresentationAttributes.Stroke != null)
00207                 {
00208                     target.StrokePath(box, BoxPresentationAttributes.Stroke,
      BoxPresentationAttributes.LineWidth, BoxPresentationAttributes.LineCap,
      BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, boxStrokeTag);
00209                     target.StrokePath(median, BoxPresentationAttributes.Stroke,
      BoxPresentationAttributes.LineWidth * 2, BoxPresentationAttributes.LineCap,
      BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, medianTag);
00210                 }
00211             }
00212             else
00213             {
00214                 GraphicsPath whiskers = new GraphicsPath();
00215
00216                 whiskers.MoveTo(whisker1Left).LineTo(whisker1Right);
00217                 whiskers.MoveTo(whisker1).LineTo(whisker2);
00218                 whiskers.MoveTo(whisker2Left).LineTo(whisker2Right);
00219
00220                 GraphicsPath box = new
      GraphicsPath().MoveTo(box1Left).LineTo(box1Right).LineTo(box2Right).LineTo(box2Left).Close();
00221                 GraphicsPath median = new GraphicsPath().MoveTo(medianLeft).LineTo(medianRight);
00222
00223                 if (WhiskersPresentationAttributes.Stroke != null)
00224                 {
00225                     target.StrokePath(whiskers, WhiskersPresentationAttributes.Stroke,
      WhiskersPresentationAttributes.LineWidth, WhiskersPresentationAttributes.LineCap,
      WhiskersPresentationAttributes.LineJoin, WhiskersPresentationAttributes.LineDash, whiskerTag);
00226                 }
00227
00228                 if (BoxPresentationAttributes.Fill != null)
00229                 {
00230                     target.FillPath(box, BoxPresentationAttributes.Fill, boxFillTag);
00231                 }
00232
00233                 if (BoxPresentationAttributes.Stroke != null)
00234                 {
00235                     target.StrokePath(box, BoxPresentationAttributes.Stroke,
      BoxPresentationAttributes.LineWidth, BoxPresentationAttributes.LineCap,
      BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, boxStrokeTag);
00236                     target.StrokePath(median, BoxPresentationAttributes.Stroke,
      BoxPresentationAttributes.LineWidth * 2, LineCaps.Butt, BoxPresentationAttributes.LineJoin,
      BoxPresentationAttributes.LineDash, medianTag);
00237                 }
00238             }
00239
00240             if (CentreSymbolPresentationAttributes.Fill != null ||
      CentreSymbolPresentationAttributes.Stroke != null)
```

```
00241             {
00242                 double medianCircleRadius = Math.Min(Math.Sqrt((medianLeft.X - medianRight.X) *
      (medianLeft.X - medianRight.X) + (medianLeft.Y - medianRight.Y) * (medianLeft.Y - medianRight.Y)),
00243                 Math.Min(Math.Sqrt((box1Left.X - box2Left.X) * (box1Left.X - box2Left.X) + (box1Left.Y
      - box2Left.Y) * (box1Left.Y - box2Left.Y)),
00244                 Math.Sqrt((box1Right.X - box2Right.X) * (box1Right.X - box2Right.X) + (box1Right.Y -
      box2Right.Y) * (box1Right.Y - box2Right.Y)))) * 0.45;
00245
00246                 target.Save();
00247                 target.Translate((medianLeft.X + medianRight.X) * 0.5, (medianLeft.Y + medianRight.Y)
      * 0.5);
00248                 target.Scale(medianCircleRadius, medianCircleRadius);
00249
00250                 CentreSymbol.Plot(target, CentreSymbolPresentationAttributes, medianSymbolTag);
00251
00252                 target.Restore();
00253             }
00254         }
00255     }
00256 }
```

8.24 CoordinateSystems.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// Represents a coordinate system.
00025 /// </summary>
00026     public interface ICoordinateSystem
00027     {
00028
00029     }
00030
00031 /// <summary>
00032 /// Represents a coordinate system transforming data points of type <typeparamref name="T"/> into plot
      <see cref="Point"/>s.
00033 /// </summary>
00034 /// <typeparam name="T">The type of data elements.</typeparam>
00035     public interface ICoordinateSystem<T> :  ICoordinateSystem
00036     {
00037 /// <summary>
00038 /// Transform the specified <paramref name="dataPoint"/> into a plot <see cref="Point"/>.
00039 /// </summary>
00040 /// <param name="dataPoint">The data point whose coordinates should be determined.</param>
00041 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00042         Point ToPlotCoordinates(T dataPoint);
00043     }
00044
00045 /// <summary>
00046 /// Represents a coordinate system tranforming data points of type <typeparamref name="T"/> into <see
      langword="double"/>s.
00047 /// </summary>
00048 /// <typeparam name="T">The type of data elements.</typeparam>
00049     public interface ICoordinateSystem1D<T>
00050     {
00051 /// <summary>
00052 /// Transform the specified <paramref name="dataPoint"/> into a <see langword="double"/>.
00053 /// </summary>
00054 /// <param name="dataPoint">The data point whose coordinates should be determined.</param>
00055 /// <returns>A <see langword="double"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00056         double ToPlotCoordinates(T dataPoint);
```

```
00057     }
00058
00059 /// <summary>
00060 /// A coordinate system using a custom method to transform data points.
00061 /// </summary>
00062 /// <typeparam name="T">The type of data elements.</typeparam>
00063     public class CoordinateSystem<T> :  ICoordinateSystem<T>
00064     {
00065 /// <summary>
00066 /// The method used to transform the data elements into plot <see cref="Point"/>s.
00067 /// </summary>
00068         public Func<T, Point> CoordinateFunction { get; set; }
00069
00070 /// <summary>
00071 /// Create a new <see cref="CoordinateSystem{T}"/> using the specified method.
00072 /// </summary>
00073 /// <param name="coordinateFunction">The method used to transform the data elements into plot <see
      cref="Point"/>s.</param>
00074         public CoordinateSystem(Func<T, Point> coordinateFunction)
00075         {
00076             this.CoordinateFunction = coordinateFunction;
00077         }
00078
00079 /// <inheritdoc/>
00080         public Point ToPlotCoordinates(T dataPoint)
00081         {
00082             return CoordinateFunction(dataPoint);
00083         }
00084     }
00085
00086 /// <summary>
00087 /// Represents a coordinate system performing continuous transformations.
00088 /// </summary>
00089     public interface IContinuousCoordinateSystem :  ICoordinateSystem<IReadOnlyList<double»
00090     {
00091 /// <summary>
00092 /// Gets whether the current coordinate system is linear along all directions.
00093 /// </summary>
00094         bool IsLinear { get; }
00095
00096 /// <summary>
00097 /// Determines whether points aligned along <paramref name="direction"/> in data space are also
      aligned along some line in plot space.
00098 /// </summary>
00099 /// <param name="direction">The direction in data space along which the points should be
      aligned.</param>
00100 /// <returns><see langword="true"/> if any two points aligned along <paramref name="direction"/> in
      data space are also aligned along some line in plot space, <see langword="false"/>
      otherwise.</returns>
00101         bool IsDirectionStraight(IReadOnlyList<double> direction);
00102
00103 /// <summary>
00104 /// The maximum difference between two points in data space that appear arbitrarily close in plot
      space, or some approximation.
00105 /// </summary>
00106         double[] Resolution { get; }
00107
00108 /// <summary>
00109 /// Gets a data element that is arbitrarily close to the specified <paramref name="point"/>, along the
      specified <paramref name="direction"/>.
00110 /// </summary>
00111 /// <param name="point">The point close to which the returned data element should be.</param>
00112 /// <param name="direction">The direction (in data space) along which the returned point should
      be.</param>
00113 /// <returns>A data element that is arbitrarily close to the specified <paramref name="point"/>, along
      the specified <paramref name="direction"/>.</returns>
00114         double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction);
00115     }
00116
00117 /// <summary>
00118 /// Represents a coordinate system performing continuous invertible transformations.
00119 /// </summary>
00120     public interface IContinuousInvertibleCoordinateSystem :  IContinuousCoordinateSystem
00121     {
00122 /// <summary>
00123 /// Transform a point in plot space back into data space.
00124 /// </summary>
00125 /// <param name="plotPoint">The point in plot space.</param>
00126 /// <returns>The point in data space corresponding to the specified point in plot space.</returns>
00127         double[] ToDataCoordinates(Point plotPoint);
00128     }
00129
00130 /// <summary>
00131 /// Represents a linear coordinate system.
00132 /// </summary>
00133     public class LinearCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00134     {
```

```
00135 /// <summary>
00136 /// The minimum X value.
00137 /// </summary>
00138         public double MinX { get; set; }
00139 
00140 /// <summary>
00141 /// The maximum X value.
00142 /// </summary>
00143         public double MaxX { get; set; }
00144 
00145 /// <summary>
00146 /// The minimum Y value.
00147 /// </summary>
00148         public double MinY { get; set; }
00149 
00150 /// <summary>
00151 /// The maximum Y value.
00152 /// </summary>
00153         public double MaxY { get; set; }
00154 
00155 /// <summary>
00156 /// The X scale.
00157 /// </summary>
00158         public double ScaleX { get; set; }
00159 
00160 /// <summary>
00161 /// The y scale.
00162 /// </summary>
00163         public double ScaleY { get; set; }
00164 
00165 /// <inheritdoc/>
00166         public bool IsLinear => true;
00167 
00168 /// <inheritdoc/>
00169         public bool IsDirectionStraight(IReadOnlyList<double> direction) => true;
00170 
00171         private double[] resolution = null;
00172 
00173 /// <inheritdoc/>
00174         public double[] Resolution
00175         {
00176             get
00177             {
00178                 return new double[] { (MaxX - MinX) * 0.01, (MaxY - MinY) * 0.01 };
00179             }
00180 
00181             set
00182             {
00183                 resolution = value;
00184             }
00185         }
00186 
00187 /// <summary>
00188 /// Creates a new <see cref="LinearCoordinateSystem2D"/> manually specifying the parameter values.
00189 /// </summary>
00190 /// <param name="minX">The minimum X value.</param>
00191 /// <param name="maxX">The maximum X value.</param>
00192 /// <param name="minY">The minimum Y value.</param>
00193 /// <param name="maxY">The maximum Y value.</param>
00194 /// <param name="scaleX">The X scale.</param>
00195 /// <param name="scaleY">The Y scale.</param>
00196         public LinearCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
      scaleX, double scaleY)
00197         {
00198             MinX = minX;
00199             MaxX = maxX;
00200             MinY = minY;
00201             MaxY = maxY;
00202             ScaleX = scaleX;
00203             ScaleY = scaleY;
00204         }
00205 
00206 /// <summary>
00207 /// Creates a new <see cref="LinearCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00208 /// </summary>
00209 /// <param name="data">The data from which the value range should be determined.</param>
00210 /// <param name="scaleX">The X scale.</param>
00211 /// <param name="scaleY">The Y scale.</param>
00212         public LinearCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double» data, double scaleX = 350,
      double scaleY = 250)
00213         {
00214             MinX = double.MaxValue;
00215             MinY = double.MaxValue;
00216 
00217             MaxX = double.MinValue;
00218             MaxY = double.MinValue;
```

```
00219
00220             for (int i = 0; i < data.Count; i++)
00221             {
00222                 MinX = Math.Min(MinX, data[i][0]);
00223                 MinY = Math.Min(MinY, data[i][1]);
00224
00225                 MaxX = Math.Max(MaxX, data[i][0]);
00226                 MaxY = Math.Max(MaxY, data[i][1]);
00227             }
00228
00229             double rangeX = MaxX - MinX;
00230             double rangeY = MaxY - MinY;
00231
00232             MinX -= rangeX * 0.1;
00233             MaxX += rangeX * 0.1;
00234
00235             MinY -= rangeY * 0.1;
00236             MaxY += rangeY * 0.1;
00237
00238             ScaleX = scaleX;
00239             ScaleY = scaleY;
00240         }
00241
00242 /// <summary>
00243 /// Creates a new <see cref="LinearCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00244 /// </summary>
00245 /// <param name="data">The data from which the value range should be determined.</param>
00246 /// <param name="scaleX">The X scale.</param>
00247 /// <param name="scaleY">The Y scale.</param>
00248         public LinearCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00249         {
00250             MinX = double.MaxValue;
00251             MinY = double.MaxValue;
00252
00253             MaxX = double.MinValue;
00254             MaxY = double.MinValue;
00255
00256             for (int i = 0; i < data.GetLength(0); i++)
00257             {
00258                 MinX = Math.Min(MinX, data[i, 0]);
00259                 MinY = Math.Min(MinY, data[i, 1]);
00260
00261                 MaxX = Math.Max(MaxX, data[i, 0]);
00262                 MaxY = Math.Max(MaxY, data[i, 1]);
00263             }
00264
00265             double rangeX = MaxX - MinX;
00266             double rangeY = MaxY - MinY;
00267
00268             MinX -= rangeX * 0.1;
00269             MaxX += rangeX * 0.1;
00270
00271             MinY -= rangeY * 0.1;
00272             MaxY += rangeY * 0.1;
00273
00274             ScaleX = scaleX;
00275             ScaleY = scaleY;
00276         }
00277
00278 /// <inheritdoc/>
00279         public double[] ToDataCoordinates(Point plotPoint)
00280         {
00281             return new double[] { plotPoint.X / ScaleX * (MaxX - MinX) + MinX, (ScaleY - plotPoint.Y)
      / ScaleY * (MaxY - MinY) + MinY };
00282         }
00283
00284 /// <inheritdoc/>
00285         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00286         {
00287             return new Point((dataPoint[0] - MinX) / (MaxX - MinX) * ScaleX, ScaleY - (dataPoint[1] -
      MinY) / (MaxY - MinY) * ScaleY);
00288         }
00289
00290 /// <summary>
00291 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00292 /// </summary>
00293 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00294 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00295         public Point ToPlotCoordinates(Point dataPoint)
00296         {
00297             return new Point((dataPoint.X - MinX) / (MaxX - MinX) * ScaleX, ScaleY - (dataPoint.Y -
      MinY) / (MaxY - MinY) * ScaleY);
00298         }
00299
00300 /// <inheritdoc/>
```

```
00301         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00302         {
00303             return new double[] { point[0] + direction[0] * this.Resolution[0], point[1] +
      direction[1] * this.Resolution[1] };
00304         }
00305     }
00306
00307 /// <summary>
00308 /// Represents a logarithmic coordinate system.
00309 /// </summary>
00310     public class LogarithmicCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00311     {
00312 /// <summary>
00313 /// The minimum X value (in logarithmic scale).
00314 /// </summary>
00315         public double MinX { get; set; }
00316
00317 /// <summary>
00318 /// The maximum X value (in logarithmic scale).
00319 /// </summary>
00320         public double MaxX { get; set; }
00321
00322 /// <summary>
00323 /// The minimum Y value (in logarithmic scale).
00324 /// </summary>
00325         public double MinY { get; set; }
00326
00327 /// <summary>
00328 /// The maximum Y value (in logarithmic scale).
00329 /// </summary>
00330         public double MaxY { get; set; }
00331
00332 /// <summary>
00333 /// The X scale.
00334 /// </summary>
00335         public double ScaleX { get; set; }
00336
00337 /// <summary>
00338 /// The y scale.
00339 /// </summary>
00340         public double ScaleY { get; set; }
00341
00342 /// <inheritdoc/>
00343         public bool IsLinear => false;
00344
00345 /// <inheritdoc/>
00346         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00347         {
00348             if (direction[0] == 0 || direction[1] == 0)
00349             {
00350                 return true;
00351             }
00352             else
00353             {
00354                 return false;
00355             }
00356         }
00357
00358         private double[] resolution = null;
00359
00360 /// <inheritdoc/>
00361         public double[] Resolution
00362         {
00363             get
00364             {
00365                 if (resolution == null)
00366                 {
00367                     return new double[] { (Math.Exp(MaxX) - Math.Exp(MinX)) * 0.01, (Math.Exp(MaxY) -
      Math.Exp(MinY)) * 0.01 };
00368                 }
00369                 else
00370                 {
00371                     return resolution;
00372                 }
00373             }
00374
00375             set
00376             {
00377                 resolution = value;
00378             }
00379         }
00380
00381 /// <summary>
00382 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> manually specifying the parameter
      values.
00383 /// </summary>
00384 /// <param name="minX">The minimum X value.</param>
```

```
00385 /// <param name="maxX">The maximum X value.</param>
00386 /// <param name="minY">The minimum Y value.</param>
00387 /// <param name="maxY">The maximum Y value.</param>
00388 /// <param name="scaleX">The X scale.</param>
00389 /// <param name="scaleY">The Y scale.</param>
00390         public LogarithmicCoordinateSystem2D(double minX, double maxX, double minY, double maxY,
      double scaleX, double scaleY)
00391         {
00392             MinX = Math.Log(minX);
00393             MaxX = Math.Log(maxX);
00394             MinY = Math.Log(minY);
00395             MaxY = Math.Log(maxY);
00396             ScaleX = scaleX;
00397             ScaleY = scaleY;
00398         }
00399 
00400 /// <summary>
00401 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00402 /// </summary>
00403 /// <param name="data">The data from which the value range should be determined.</param>
00404 /// <param name="scaleX">The X scale.</param>
00405 /// <param name="scaleY">The Y scale.</param>
00406         public LogarithmicCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double» data, double scaleX =
      350, double scaleY = 250)
00407         {
00408             MinX = double.MaxValue;
00409             MinY = double.MaxValue;
00410 
00411             MaxX = double.MinValue;
00412             MaxY = double.MinValue;
00413 
00414             for (int i = 0; i < data.Count; i++)
00415             {
00416                 MinX = Math.Min(MinX, data[i][0]);
00417                 MinY = Math.Min(MinY, data[i][1]);
00418 
00419                 MaxX = Math.Max(MaxX, data[i][0]);
00420                 MaxY = Math.Max(MaxY, data[i][1]);
00421             }
00422 
00423             MinX = Math.Log(MinX);
00424             MinY = Math.Log(MinY);
00425             MaxX = Math.Log(MaxX);
00426             MaxY = Math.Log(MaxY);
00427 
00428             double rangeX = MaxX - MinX;
00429             double rangeY = MaxY - MinY;
00430 
00431             MinX -= rangeX * 0.1;
00432             MaxX += rangeX * 0.1;
00433 
00434             MinY -= rangeY * 0.1;
00435             MaxY += rangeY * 0.1;
00436 
00437             ScaleX = scaleX;
00438             ScaleY = scaleY;
00439         }
00440 
00441 
00442 /// <summary>
00443 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00444 /// </summary>
00445 /// <param name="data">The data from which the value range should be determined.</param>
00446 /// <param name="scaleX">The X scale.</param>
00447 /// <param name="scaleY">The Y scale.</param>
00448         public LogarithmicCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00449         {
00450             MinX = double.MaxValue;
00451             MinY = double.MaxValue;
00452 
00453             MaxX = double.MinValue;
00454             MaxY = double.MinValue;
00455 
00456             for (int i = 0; i < data.GetLength(0); i++)
00457             {
00458                 MinX = Math.Min(MinX, data[i, 0]);
00459                 MinY = Math.Min(MinY, data[i, 1]);
00460 
00461                 MaxX = Math.Max(MaxX, data[i, 0]);
00462                 MaxY = Math.Max(MaxY, data[i, 1]);
00463             }
00464 
00465             MinX = Math.Log(MinX);
00466             MinY = Math.Log(MinY);
00467             MaxX = Math.Log(MaxX);
```

```
00468             MaxY = Math.Log(MaxY);
00469
00470             double rangeX = MaxX - MinX;
00471             double rangeY = MaxY - MinY;
00472
00473             MinX -= rangeX * 0.1;
00474             MaxX += rangeX * 0.1;
00475
00476             MinY -= rangeY * 0.1;
00477             MaxY += rangeY * 0.1;
00478
00479             ScaleX = scaleX;
00480             ScaleY = scaleY;
00481         }
00482
00483 /// <inheritdoc/>
00484         public double[] ToDataCoordinates(Point plotPoint)
00485         {
00486             return new double[] { Math.Exp(plotPoint.X / ScaleX * (MaxX - MinX) + MinX),
      Math.Exp((ScaleY - plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY) };
00487         }
00488
00489 /// <inheritdoc/>
00490         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00491         {
00492             return new Point((Math.Log(dataPoint[0]) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint[1]) - MinY) / (MaxY - MinY) * ScaleY);
00493         }
00494
00495 /// <summary>
00496 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00497 /// </summary>
00498 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00499 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00500         public Point ToPlotCoordinates(Point dataPoint)
00501         {
00502             return new Point((Math.Log(dataPoint.X) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint.Y) - MinY) / (MaxY - MinY) * ScaleY);
00503         }
00504
00505 /// <inheritdoc/>
00506         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00507         {
00508             return new double[] { Math.Exp(Math.Log(point[0]) + direction[0] * (MaxX - MinX) * 0.01),
      Math.Exp(Math.Log(point[1]) + direction[1] * (MaxY - MinY) * 0.01) };
00509         }
00510     }
00511
00512 /// <summary>
00513 /// Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.
00514 /// </summary>
00515     public class LogLinCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00516     {
00517 /// <summary>
00518 /// The minimum X value.
00519 /// </summary>
00520         public double MinX { get; set; }
00521
00522 /// <summary>
00523 /// The maximum X value.
00524 /// </summary>
00525         public double MaxX { get; set; }
00526
00527 /// <summary>
00528 /// The minimum Y value (in logarithmic scale).
00529 /// </summary>
00530         public double MinY { get; set; }
00531
00532 /// <summary>
00533 /// The maximum Y value (in logarithmic scale).
00534 /// </summary>
00535         public double MaxY { get; set; }
00536
00537 /// <summary>
00538 /// The X scale.
00539 /// </summary>
00540         public double ScaleX { get; set; }
00541
00542 /// <summary>
00543 /// The y scale.
00544 /// </summary>
00545         public double ScaleY { get; set; }
00546
00547 /// <inheritdoc/>
00548         public bool IsLinear => false;
00549
```

```
00550 /// <inheritdoc/>
00551         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00552         {
00553             if (direction[0] == 0 || direction[1] == 0)
00554             {
00555                 return true;
00556             }
00557             else
00558             {
00559                 return false;
00560             }
00561         }
00562 
00563         private double[] resolution = null;
00564 
00565 /// <inheritdoc/>
00566         public double[] Resolution
00567         {
00568             get
00569             {
00570                 if (resolution == null)
00571                 {
00572                     return new double[] { (MaxX - MinX) * 0.01, (Math.Exp(MaxY) - Math.Exp(MinY)) *
    0.01 };
00573                 }
00574                 else
00575                 {
00576                     return resolution;
00577                 }
00578             }
00579 
00580             set
00581             {
00582                 resolution = value;
00583             }
00584         }
00585 
00586 /// <summary>
00587 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> manually specifying the parameter values.
00588 /// </summary>
00589 /// <param name="minX">The minimum X value.</param>
00590 /// <param name="maxX">The maximum X value.</param>
00591 /// <param name="minY">The minimum Y value.</param>
00592 /// <param name="maxY">The maximum Y value.</param>
00593 /// <param name="scaleX">The X scale.</param>
00594 /// <param name="scaleY">The Y scale.</param>
00595         public LogLinCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
    scaleX, double scaleY)
00596         {
00597             MinX = minX;
00598             MaxX = maxX;
00599             MinY = Math.Log(minY);
00600             MaxY = Math.Log(maxY);
00601             ScaleX = scaleX;
00602             ScaleY = scaleY;
00603         }
00604 
00605 /// <summary>
00606 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> determining the value range from the
    <paramref name="data"/>.
00607 /// </summary>
00608 /// <param name="data">The data from which the value range should be determined.</param>
00609 /// <param name="scaleX">The X scale.</param>
00610 /// <param name="scaleY">The Y scale.</param>
00611         public LogLinCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double>> data, double scaleX = 350,
    double scaleY = 250)
00612         {
00613             MinX = double.MaxValue;
00614             MinY = double.MaxValue;
00615 
00616             MaxX = double.MinValue;
00617             MaxY = double.MinValue;
00618 
00619             for (int i = 0; i < data.Count; i++)
00620             {
00621                 MinX = Math.Min(MinX, data[i][0]);
00622                 MinY = Math.Min(MinY, data[i][1]);
00623 
00624                 MaxX = Math.Max(MaxX, data[i][0]);
00625                 MaxY = Math.Max(MaxY, data[i][1]);
00626             }
00627 
00628             MinX = MinX;
00629             MinY = Math.Log(MinY);
00630             MaxX = MaxX;
00631             MaxY = Math.Log(MaxY);
00632 
```

```
00633             double rangeX = MaxX - MinX;
00634             double rangeY = MaxY - MinY;
00635 
00636             MinX -= rangeX * 0.1;
00637             MaxX += rangeX * 0.1;
00638 
00639             MinY -= rangeY * 0.1;
00640             MaxY += rangeY * 0.1;
00641 
00642             ScaleX = scaleX;
00643             ScaleY = scaleY;
00644         }
00645 
00646 /// <summary>
00647 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00648 /// </summary>
00649 /// <param name="data">The data from which the value range should be determined.</param>
00650 /// <param name="scaleX">The X scale.</param>
00651 /// <param name="scaleY">The Y scale.</param>
00652         public LogLinCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00653         {
00654             MinX = double.MaxValue;
00655             MinY = double.MaxValue;
00656 
00657             MaxX = double.MinValue;
00658             MaxY = double.MinValue;
00659 
00660             for (int i = 0; i < data.GetLength(0); i++)
00661             {
00662                 MinX = Math.Min(MinX, data[i, 0]);
00663                 MinY = Math.Min(MinY, data[i, 1]);
00664 
00665                 MaxX = Math.Max(MaxX, data[i, 0]);
00666                 MaxY = Math.Max(MaxY, data[i, 1]);
00667             }
00668 
00669             MinX = MinX;
00670             MinY = Math.Log(MinY);
00671             MaxX = MaxX;
00672             MaxY = Math.Log(MaxY);
00673 
00674             double rangeX = MaxX - MinX;
00675             double rangeY = MaxY - MinY;
00676 
00677             MinX -= rangeX * 0.1;
00678             MaxX += rangeX * 0.1;
00679 
00680             MinY -= rangeY * 0.1;
00681             MaxY += rangeY * 0.1;
00682 
00683             ScaleX = scaleX;
00684             ScaleY = scaleY;
00685         }
00686 
00687 /// <inheritdoc/>
00688         public double[] ToDataCoordinates(Point plotPoint)
00689         {
00690             return new double[] { plotPoint.X / ScaleX * (MaxX - MinX) + MinX, Math.Exp((ScaleY -
      plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY) };
00691         }
00692 
00693 /// <inheritdoc/>
00694         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00695         {
00696             return new Point((dataPoint[0] - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint[1]) - MinY) / (MaxY - MinY) * ScaleY);
00697         }
00698 
00699 /// <summary>
00700 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00701 /// </summary>
00702 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00703 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00704         public Point ToPlotCoordinates(Point dataPoint)
00705         {
00706             return new Point((dataPoint.X - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint.Y) - MinY) / (MaxY - MinY) * ScaleY);
00707         }
00708 
00709 /// <inheritdoc/>
00710         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00711         {
00712             return new double[] { point[0] + direction[0] * (MaxX - MinX) * 0.01,
      Math.Exp(Math.Log(point[1]) + direction[1] * (MaxY - MinY) * 0.01) };
00713         }
```

```
00714     }
00715
00716 /// <summary>
00717 /// Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.
00718 /// </summary>
00719     public class LinLogCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00720     {
00721 /// <summary>
00722 /// The minimum X value.
00723 /// </summary>
00724         public double MinX { get; set; }
00725
00726 /// <summary>
00727 /// The maximum X value.
00728 /// </summary>
00729         public double MaxX { get; set; }
00730
00731 /// <summary>
00732 /// The minimum Y value (in logarithmic scale).
00733 /// </summary>
00734         public double MinY { get; set; }
00735
00736 /// <summary>
00737 /// The maximum Y value (in logarithmic scale).
00738 /// </summary>
00739         public double MaxY { get; set; }
00740
00741 /// <summary>
00742 /// The X scale.
00743 /// </summary>
00744         public double ScaleX { get; set; }
00745
00746 /// <summary>
00747 /// The y scale.
00748 /// </summary>
00749         public double ScaleY { get; set; }
00750
00751 /// <inheritdoc/>
00752         public bool IsLinear => false;
00753
00754 /// <inheritdoc/>
00755         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00756         {
00757             if (direction[0] == 0 || direction[1] == 0)
00758             {
00759                 return true;
00760             }
00761             else
00762             {
00763                 return false;
00764             }
00765         }
00766
00767         private double[] resolution = null;
00768
00769 /// <inheritdoc/>
00770         public double[] Resolution
00771         {
00772             get
00773             {
00774                 if (resolution == null)
00775                 {
00776                     return new double[] { (MaxX - MinX) * 0.01, (Math.Exp(MaxY) - Math.Exp(MinY)) *
      0.01 };
00777                 }
00778                 else
00779                 {
00780                     return resolution;
00781                 }
00782             }
00783
00784             set
00785             {
00786                 resolution = value;
00787             }
00788         }
00789
00790 /// <summary>
00791 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> manually specifying the parameter values.
00792 /// </summary>
00793 /// <param name="minX">The minimum X value.</param>
00794 /// <param name="maxX">The maximum X value.</param>
00795 /// <param name="minY">The minimum Y value.</param>
00796 /// <param name="maxY">The maximum Y value.</param>
00797 /// <param name="scaleX">The X scale.</param>
00798 /// <param name="scaleY">The Y scale.</param>
00799         public LinLogCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
```

```
      scaleX, double scaleY)
00800         {
00801             MinX = Math.Log(minX);
00802             MaxX = Math.Log(maxX);
00803             MinY = minY;
00804             MaxY = maxY;
00805             ScaleX = scaleX;
00806             ScaleY = scaleY;
00807         }
00808 
00809 /// <summary>
00810 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00811 /// </summary>
00812 /// <param name="data">The data from which the value range should be determined.</param>
00813 /// <param name="scaleX">The X scale.</param>
00814 /// <param name="scaleY">The Y scale.</param>
00815         public LinLogCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double> data, double scaleX = 350,
      double scaleY = 250)
00816         {
00817             MinX = double.MaxValue;
00818             MinY = double.MaxValue;
00819 
00820             MaxX = double.MinValue;
00821             MaxY = double.MinValue;
00822 
00823             for (int i = 0; i < data.Count; i++)
00824             {
00825                 MinX = Math.Min(MinX, data[i][0]);
00826                 MinY = Math.Min(MinY, data[i][1]);
00827 
00828                 MaxX = Math.Max(MaxX, data[i][0]);
00829                 MaxY = Math.Max(MaxY, data[i][1]);
00830             }
00831 
00832             MinX = Math.Log(MinX);
00833             MinY = MinY;
00834             MaxX = Math.Log(MaxX);
00835             MaxY = MaxY;
00836 
00837             double rangeX = MaxX - MinX;
00838             double rangeY = MaxY - MinY;
00839 
00840             MinX -= rangeX * 0.1;
00841             MaxX += rangeX * 0.1;
00842 
00843             MinY -= rangeY * 0.1;
00844             MaxY += rangeY * 0.1;
00845 
00846             ScaleX = scaleX;
00847             ScaleY = scaleY;
00848         }
00849 
00850 /// <summary>
00851 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00852 /// </summary>
00853 /// <param name="data">The data from which the value range should be determined.</param>
00854 /// <param name="scaleX">The X scale.</param>
00855 /// <param name="scaleY">The Y scale.</param>
00856         public LinLogCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00857         {
00858             MinX = double.MaxValue;
00859             MinY = double.MaxValue;
00860 
00861             MaxX = double.MinValue;
00862             MaxY = double.MinValue;
00863 
00864             for (int i = 0; i < data.GetLength(0); i++)
00865             {
00866                 MinX = Math.Min(MinX, data[i, 0]);
00867                 MinY = Math.Min(MinY, data[i, 1]);
00868 
00869                 MaxX = Math.Max(MaxX, data[i, 0]);
00870                 MaxY = Math.Max(MaxY, data[i, 1]);
00871             }
00872 
00873             MinX = Math.Log(MinX);
00874             MinY = MinY;
00875             MaxX = Math.Log(MaxX);
00876             MaxY = MaxY;
00877 
00878             double rangeX = MaxX - MinX;
00879             double rangeY = MaxY - MinY;
00880 
00881             MinX -= rangeX * 0.1;
00882             MaxX += rangeX * 0.1;
```

```
00883
00884             MinY -= rangeY * 0.1;
00885             MaxY += rangeY * 0.1;
00886
00887             ScaleX = scaleX;
00888             ScaleY = scaleY;
00889         }
00890
00891 /// <inheritdoc/>
00892         public double[] ToDataCoordinates(Point plotPoint)
00893         {
00894             return new double[] { Math.Exp(plotPoint.X / ScaleX * (MaxX - MinX) + MinX), (ScaleY -
      plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY };
00895         }
00896
00897 /// <inheritdoc/>
00898         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00899         {
00900             return new Point((Math.Log(dataPoint[0]) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (dataPoint[1] - MinY) / (MaxY - MinY) * ScaleY);
00901         }
00902
00903 /// <summary>
00904 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00905 /// </summary>
00906 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00907 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00908         public Point ToPlotCoordinates(Point dataPoint)
00909         {
00910             return new Point((Math.Log(dataPoint.X) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (dataPoint.Y - MinY) / (MaxY - MinY) * ScaleY);
00911         }
00912
00913 /// <inheritdoc/>
00914         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00915         {
00916             return new double[] { Math.Exp(Math.Log(point[0]) + direction[0] * (MaxX - MinX) * 0.01),
      point[1] + direction[1] * (MaxY - MinY) * 0.01 };
00917         }
00918     }
00919
00920 /// <summary>
00921 /// Represents a 1-D linear coordinate system.
00922 /// </summary>
00923     public class LinearCoordinateSystem1D :  ICoordinateSystem1D<double>
00924     {
00925 /// <summary>
00926 /// The minimum value.
00927 /// </summary>
00928         public double Min { get; set; }
00929
00930 /// <summary>
00931 /// The maximum value.
00932 /// </summary>
00933         public double Max { get; set; }
00934
00935 /// <summary>
00936 /// The scale factor.
00937 /// </summary>
00938         public double Scale { get; set; }
00939
00940 /// <summary>
00941 /// Creates a new <see cref="LinearCoordinateSystem1D"/>, manually specifying the parameters.
00942 /// </summary>
00943 /// <param name="min">The minimum value.</param>
00944 /// <param name="max">The maximum value.</param>
00945 /// <param name="scale">The scale factor.</param>
00946         public LinearCoordinateSystem1D(double min, double max, double scale)
00947         {
00948             Min = min;
00949             Max = max;
00950             Scale = scale;
00951         }
00952
00953 /// <summary>
00954 /// Creates a new <see cref="LinearCoordinateSystem1D"/> determining the value range from the
      <paramref name="data"/>.
00955 /// </summary>
00956 /// <param name="data">The data from which the value range should be determined.</param>
00957 /// <param name="scale">The scale factor.</param>
00958         public LinearCoordinateSystem1D(IReadOnlyList<double> data, double scale = 350)
00959         {
00960             Min = double.MaxValue;
00961             Min = double.MaxValue;
00962
00963             for (int i = 0; i < data.Count; i++)
```

```
00964             {
00965                 Min = Math.Min(Min, data[i]);
00966                 Max = Math.Max(Max, data[i]);
00967             }
00968 
00969             double range = Max - Min;
00970 
00971             Min -= range * 0.1;
00972             Max += range * 0.1;
00973 
00974             Scale = scale;
00975         }
00976 
00977 /// <inheritdoc/>
00978         public double ToPlotCoordinates(double dataPoint)
00979         {
00980             return (dataPoint - Min) / (Max - Min) * Scale;
00981         }
00982     }
00983 
00984 /// <summary>
00985 /// Represents a 1-D logarithmic coordinate system.
00986 /// </summary>
00987     public class LogarithmicCoordinateSystem1D :  ICoordinateSystem1D<double>
00988     {
00989 /// <summary>
00990 /// The minimum value (in logarithmic scale).
00991 /// </summary>
00992         public double Min { get; set; }
00993 
00994 /// <summary>
00995 /// The maximum value (in logarithmic scale).
00996 /// </summary>
00997         public double Max { get; set; }
00998 
00999 /// <summary>
01000 /// The scale factor.
01001 /// </summary>
01002         public double Scale { get; set; }
01003 
01004 /// <summary>
01005 /// Creates a new <see cref="LogarithmicCoordinateSystem1D"/>, manually specifying the parameters.
01006 /// </summary>
01007 /// <param name="min">The minimum value.</param>
01008 /// <param name="max">The maximum value.</param>
01009 /// <param name="scale">The scale factor.</param>
01010         public LogarithmicCoordinateSystem1D(double min, double max, double scale)
01011         {
01012             Min = Math.Log(min);
01013             Max = Math.Log(max);
01014             Scale = scale;
01015         }
01016 
01017 /// <summary>
01018 /// Creates a new <see cref="LogarithmicCoordinateSystem1D"/> determining the value range from the
      <paramref name="data"/>.
01019 /// </summary>
01020 /// <param name="data">The data from which the value range should be determined.</param>
01021 /// <param name="scale">The scale factor.</param>
01022         public LogarithmicCoordinateSystem1D(IReadOnlyList<double> data, double scale = 350)
01023         {
01024             Min = double.MaxValue;
01025             Min = double.MaxValue;
01026 
01027             for (int i = 0; i < data.Count; i++)
01028             {
01029                 Min = Math.Min(Min, data[i]);
01030                 Max = Math.Max(Max, data[i]);
01031             }
01032 
01033             Min = Math.Log(Min);
01034             Max = Math.Log(Max);
01035 
01036             double range = Max - Min;
01037 
01038             Min -= range * 0.1;
01039             Max += range * 0.1;
01040 
01041             Scale = scale;
01042         }
01043 
01044 /// <inheritdoc/>
01045         public double ToPlotCoordinates(double dataPoint)
01046         {
01047             return (Math.Log(dataPoint) - Min) / (Max - Min) * Scale;
01048         }
01049     }
```

```
01050
01051 /// <summary>
01052 /// A coordinate system using a custom method to transform data points.
01053 /// </summary>
01054 /// <typeparam name="T">The type of data elements.</typeparam>
01055     public class CoordinateSystem1D<T> :  ICoordinateSystem1D<T>
01056     {
01057 /// <summary>
01058 /// The method used to transform data points.
01059 /// </summary>
01060         public Func<T, double> CoordinateFunction { get; set; }
01061
01062 /// <summary>
01063 /// Creates a new <see cref="CoordinateSystem1D{T}"/> using the specified <paramref
      name="coordinateFunction"/>.
01064 /// </summary>
01065 /// <param name="coordinateFunction">The method used to transform data points.</param>
01066         public CoordinateSystem1D(Func<T, double> coordinateFunction)
01067         {
01068             this.CoordinateFunction = coordinateFunction;
01069         }
01070
01071 /// <inheritdoc/>
01072         public double ToPlotCoordinates(T dataPoint)
01073         {
01074             return CoordinateFunction(dataPoint);
01075         }
01076     }
01077
01078 /// <summary>
01079 /// Represents a categorical 1-D coordinate system.
01080 /// </summary>
01081 /// <typeparam name="T">The type of data elements.</typeparam>
01082     public class CategoricalCoordinateSystem1D<T> :  ICoordinateSystem1D<T>
01083     {
01084 /// <summary>
01085 /// A <see cref="Dictionary{TKey, TValue}"/> storing the coordinates.
01086 /// </summary>
01087         public Dictionary<T, double> Coordinates { get; set; }
01088
01089 /// <summary>
01090 /// Creates a new <see cref="CategoricalCoordinateSystem1D{T}"/> using the coordinates specified in
      the supplied <paramref name="coordinates"/> <see cref="Dictionary{TKey, TValue}"/>.
01091 /// </summary>
01092 /// <param name="coordinates"></param>
01093         public CategoricalCoordinateSystem1D(Dictionary<T, double> coordinates)
01094         {
01095             this.Coordinates = coordinates;
01096         }
01097
01098 /// <summary>
01099 /// Creates a new <see cref="CategoricalCoordinateSystem1D{T}"/> assigning equally-spaced coordinates
      to all different values of <paramref name="data"/>.
01100 /// </summary>
01101 /// <param name="data">The data containing discrete parameter values that should be assigned to
      different coordinates.</param>
01102 /// <param name="scale">The maximum coordinate.</param>
01103         public CategoricalCoordinateSystem1D(IReadOnlyList<T> data, double scale = 350)
01104         {
01105             Coordinates = new Dictionary<T, double>();
01106
01107             for (int i = 0; i < data.Count; i++)
01108             {
01109                 if (!Coordinates.ContainsKey(data[i]))
01110                 {
01111                     Coordinates[data[i]] = Coordinates.Count;
01112                 }
01113             }
01114
01115             if (Coordinates.Count > 1)
01116             {
01117                 foreach (KeyValuePair<T, double> kvp in Coordinates)
01118                 {
01119                     Coordinates[kvp.Key] = kvp.Value / (Coordinates.Count - 1) * scale; ;
01120                 }
01121             }
01122         }
01123
01124 /// <inheritdoc/>
01125         public double ToPlotCoordinates(T dataPoint)
01126         {
01127             return Coordinates[dataPoint];
01128         }
01129     }
01130
01131 /// <summary>
01132 /// Combines two <see cref="ICoordinateSystem1D{T}"/>s to produce a <see
```

```
      cref="ICoordinateSystem{T}"/>.
01133 /// </summary>
01134 /// <typeparam name="T1">The type for the first coordinate system.</typeparam>
01135 /// <typeparam name="T2">The type for the second coordinate system.</typeparam>
01136     public class CompositeCoordinateSystem2D<T1, T2> :  ICoordinateSystem<(T1, T2)>
01137     {
01138 /// <summary>
01139 /// The first coordinate system.
01140 /// </summary>
01141         public ICoordinateSystem1D<T1> CoordinateSystemX { get; set; }
01142 
01143 /// <summary>
01144 /// The second coordinate system.
01145 /// </summary>
01146         public ICoordinateSystem1D<T2> CoordinateSystemY { get; set; }
01147 
01148 /// <summary>
01149 /// Creates a new <see cref="CompositeCoordinateSystem2D{T1, T2}"/> from two 1-D coordinate system.
01150 /// </summary>
01151 /// <param name="coordinateSystemX">The first coordinate system.</param>
01152 /// <param name="coordinateSystemY">The second coordinate system.</param>
01153         public CompositeCoordinateSystem2D(ICoordinateSystem1D<T1> coordinateSystemX,
      ICoordinateSystem1D<T2> coordinateSystemY)
01154         {
01155             this.CoordinateSystemX = coordinateSystemX;
01156             this.CoordinateSystemY = coordinateSystemY;
01157         }
01158 
01159 /// <inheritdoc/>
01160         public Point ToPlotCoordinates((T1, T2) dataPoint)
01161         {
01162             return new Point(CoordinateSystemX.ToPlotCoordinates(dataPoint.Item1),
      CoordinateSystemY.ToPlotCoordinates(dataPoint.Item2));
01163         }
01164     }
01165 }
```

8.25 DataPoints.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// Represents a symbol that can be added to the plot at a specified position.
00025 /// </summary>
00026     public interface IDataPointElement
00027     {
00028 /// <summary>
00029 /// Draw the symbol on the plot.
00030 /// </summary>
00031 /// <param name="target">The <see cref="Graphics"/> object on which to draw.  It is assumed that it
      has been transformed so that the symbol can be drawn centred at (0, 0)</param>
00032 /// <param name="presentationAttributes">Presentation attributes determining the appearance of the
      symbol.</param>
00033 /// <param name="tag">A tag to identify the symbol in the plot.</param>
00034         void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes, string
      tag);
00035     }
00036 
00037 /// <summary>
00038 /// A symbol defined by a <see cref="GraphicsPath"/>.
00039 /// </summary>
00040     public class PathDataPointElement :  IDataPointElement
00041     {
```

```
00042 /// <summary>
00043 /// The <see cref="GraphicsPath"/> that constitutes the symbol (by default, a circle).
00044 /// </summary>
00045         public GraphicsPath Path { get; set; } = new GraphicsPath().Arc(0, 0, 1, 0, 2 *
      Math.PI).Close();
00046
00047 /// <inheritdoc/>
00048         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag)
00049         {
00050             if (string.IsNullOrEmpty(tag))
00051             {
00052                 if (presentationAttributes.Fill != null)
00053                 {
00054                     target.FillPath(Path, presentationAttributes.Fill);
00055                 }
00056
00057                 if (presentationAttributes.Stroke != null)
00058                 {
00059                     target.StrokePath(Path, presentationAttributes.Stroke,
      presentationAttributes.LineWidth, presentationAttributes.LineCap, presentationAttributes.LineJoin,
      presentationAttributes.LineDash);
00060                 }
00061             }
00062             else
00063             {
00064                 if (presentationAttributes.Fill != null)
00065                 {
00066                     target.FillPath(Path, presentationAttributes.Fill, tag);
00067                 }
00068
00069                 if (presentationAttributes.Stroke != null)
00070                 {
00071                     target.StrokePath(Path, presentationAttributes.Stroke,
      presentationAttributes.LineWidth, presentationAttributes.LineCap, presentationAttributes.LineJoin,
      presentationAttributes.LineDash, tag + (target.UseUniqueTags ?  "@stroke" :  ""));
00072                 }
00073             }
00074         }
00075
00076 /// <summary>
00077 /// Create a new <see cref="PathDataPointElement"/> instance representing a circle.
00078 /// </summary>
00079         public PathDataPointElement()
00080         {
00081
00082         }
00083
00084 /// <summary>
00085 /// Create a new <see cref="PathDataPointElement"/> instance with the specified <paramref
      name="path"/>.
00086 /// </summary>
00087         public PathDataPointElement(GraphicsPath path)
00088         {
00089             this.Path = path;
00090         }
00091     }
00092
00093 /// <summary>
00094 /// A symbol defined by a <see cref="VectSharp.Graphics"/> object.
00095 /// </summary>
00096     public class GraphicsDataPointElement :  IDataPointElement
00097     {
00098 /// <summary>
00099 /// The <see cref="VectSharp.Graphics"/> object that will be copied on the plot.
00100 /// </summary>
00101         public Graphics Graphics { get; set; }
00102
00103 /// <inheritdoc/>
00104         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag)
00105         {
00106             target.DrawGraphics(0, 0, Graphics, tag);
00107         }
00108
00109 /// <summary>
00110 /// Creates a new <see cref="GraphicsDataPointElement"/> instance.
00111 /// </summary>
00112 /// <param name="graphics"></param>
00113         public GraphicsDataPointElement(Graphics graphics)
00114         {
00115             Graphics = graphics;
00116         }
00117     }
00118
00119 /// <summary>
00120 /// A symbol drawn by a custom <see cref="Action"/>.
```

```
00121 /// </summary>
00122     public class ActionDataPointElement :  IDataPointElement
00123     {
00124 /// <summary>
00125 /// The <see cref="Action"/> used to draw the symbol.  This should take as arguments the <see
      cref="Graphics"/>
00126 /// object on which to draw the symbol, the <see cref="PlotElementPresentationAttributes"/> describing
      the
00127 /// appearance of the symbol, and a <see langword="string"/> representing a tag for the symbol.
00128 /// </summary>
00129         public Action<Graphics, PlotElementPresentationAttributes, string> PlotAction { get; set; }
00130
00131 /// <inheritdoc/>
00132         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag) => PlotAction(target, presentationAttributes, tag);
00133
00134 /// <summary>
00135 /// Creates a new <see cref="ActionDataPointElement"/> using the specified action to draw the symbol.
00136 /// </summary>
00137 /// <param name="plotAction">The <see cref="Action"/> used to draw the symbol.  This should take as
      arguments the <see cref="Graphics"/>
00138 /// object on which to draw the symbol, the <see cref="PlotElementPresentationAttributes"/> describing
      the
00139 /// appearance of the symbol, and a <see langword="string"/> representing a tag for the
      symbol.</param>
00140         public ActionDataPointElement(Action<Graphics, PlotElementPresentationAttributes, string>
      plotAction)
00141         {
00142             PlotAction = plotAction;
00143         }
00144     }
00145
00146 /// <summary>
00147 /// A plot element that draws a symbol at the location of multiple data points.
00148 /// </summary>
00149 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00150     public class ScatterPoints<T> :  IPlotElement
00151     {
00152 /// <summary>
00153 /// The data points at which the symbols will be drawn.
00154 /// </summary>
00155         public IEnumerable<T> Data { get; set; }
00156
00157 /// <summary>
00158 /// The size of the symbols to draw, in plot coordinates.
00159 /// </summary>
00160         public double Size { get; set; } = 2;
00161
00162 /// <summary>
00163 /// The symbol that will be drawn (by default, a circle).
00164 /// </summary>
00165         public IDataPointElement DataPointElement { get; set; } = new PathDataPointElement();
00166
00167 /// <summary>
00168 /// The coordinate system used to transform the points from data space to plot space.
00169 /// </summary>
00170         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00171         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00172
00173 /// <summary>
00174 /// Presentation attributes determining the appearance (stroke and fill colour, etc.)  of the symbols.
00175 /// </summary>
00176         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00177
00178 /// <summary>
00179 /// A tag to identify the symbols in the plot.
00180 /// </summary>
00181         public string Tag { get; set; }
00182
00183 /// <summary>
00184 /// Creates a new <see cref="ScatterPoints{T}"/> instance, using the specified <paramref name="data"/>
      and <paramref name="coordinateSystem"/>.
00185 /// </summary>
00186 /// <param name="data">The data points at which the symbols will be drawn.</param>
00187 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00188         public ScatterPoints(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00189         {
00190             this.Data = data;
00191             this.CoordinateSystem = coordinateSystem;
00192         }
00193
00194 /// <inheritdoc/>
00195         public void Plot(Graphics target)
00196         {
```

```
00197             int index = 0;
00198
00199             foreach (T data in Data)
00200             {
00201                 Point pt = CoordinateSystem.ToPlotCoordinates(data);
00202
00203                 target.Save();
00204                 target.Translate(pt);
00205                 target.Scale(Size, Size);
00206
00207                 if (!target.UseUniqueTags || string.IsNullOrEmpty(Tag))
00208                 {
00209                     DataPointElement.Plot(target, PresentationAttributes, Tag);
00210                 }
00211                 else
00212                 {
00213                     DataPointElement.Plot(target, PresentationAttributes, Tag + "@" +
      index.ToString());
00214                     index++;
00215                 }
00216
00217                 target.Restore();
00218             }
00219         }
00220     }
00221
00222 /// <summary>
00223 /// A plot element that draws a single text label.
00224 /// </summary>
00225 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00226     public class TextLabel<T> :  IPlotElement
00227     {
00228 /// <summary>
00229 /// The position of the label, in data space coordinates.
00230 /// </summary>
00231         public T Position { get; set; }
00232
00233 /// <summary>
00234 /// The text of the label.  This can include formatting specifiers (see the documentation for the <see
      cref="FormattedText.Format(string, Font, Font, Font, Font, Brush)"/> method).
00235 /// </summary>
00236         public string Label { get; set; }
00237
00238 /// <summary>
00239 /// The angle at which the text is drawn, with respect to the horizontal.
00240 /// </summary>
00241         public double Rotation { get; set; } = 0;
00242
00243 /// <summary>
00244 /// The baseline for the text.
00245 /// </summary>
00246         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00247
00248 /// <summary>
00249 /// The alignment for the text.
00250 /// </summary>
00251         public TextAnchors Alignment { get; set; } = TextAnchors.Center;
00252
00253 /// <summary>
00254 /// The coordinate system used to transform the points from data space to plot space.
00255 /// </summary>
00256         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00257         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00258
00259 /// <summary>
00260 /// Presentation attributes determining the appearance (stroke and fill colour, etc.)  of the text
      label.
00261 /// </summary>
00262         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Stroke = null };
00263
00264 /// <summary>
00265 /// A tag to identify the label in the plot.
00266 /// </summary>
00267         public string Tag { get; set; }
00268
00269 /// <summary>
00270 /// Create a new <see cref="TextLabel{T}"/> instance.
00271 /// </summary>
00272 /// <param name="label">The text of the label.  This can include formatting specifiers (see the
      documentation for the <see cref="FormattedText.Format(string, Font, Font, Font, Font, Brush)"/>
      method).</param>
00273 /// <param name="position">The position of the label, in data space coordinates.</param>
00274 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00275         public TextLabel(string label, T position, ICoordinateSystem<T> coordinateSystem)
```

```
00276         {
00277             this.Label = label;
00278             this.Position = position;
00279             this.CoordinateSystem = coordinateSystem;
00280         }
00281 
00282 /// <inheritdoc/>
00283         public void Plot(Graphics target)
00284         {
00285             if (!string.IsNullOrEmpty(Label))
00286             {
00287                 IEnumerable<FormattedText> label;
00288 
00289                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00290                 {
00291                     int i = Array.IndexOf(FontFamily.StandardFamilies,
    PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00292 
00293                     label = FormattedText.Format(this.Label, (FontFamily.StandardFontFamilies)i,
    PresentationAttributes.Font.FontSize);
00294                 }
00295                 else
00296                 {
00297                     label = FormattedText.Format(this.Label, PresentationAttributes.Font,
    PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00298                 }
00299 
00300                 Point point = CoordinateSystem.ToPlotCoordinates(Position);
00301 
00302                 target.Save();
00303                 target.Translate(point);
00304                 target.Rotate(Rotation);
00305 
00306                 double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
    -label.Measure().Width :  -label.Measure().Width * 0.5;
00307 
00308                 string fillTag = Tag;
00309                 string strokeTag = Tag;
00310 
00311                 if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00312                 {
00313                     strokeTag = strokeTag + "@stroke";
00314                 }
00315 
00316                 if (PresentationAttributes.Fill != null)
00317                 {
00318                     target.FillText(x, 0, label, PresentationAttributes.Fill, Baseline, fillTag);
00319                 }
00320 
00321                 if (PresentationAttributes.Stroke != null)
00322                 {
00323                     target.StrokeText(x, 0, label, PresentationAttributes.Stroke, Baseline,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
    PresentationAttributes.LineDash, strokeTag);
00324                 }
00325 
00326                 target.Restore();
00327             }
00328         }
00329 
00330     }
00331 
00332 /// <summary>
00333 /// A plot element that draws a text label at each data point.
00334 /// </summary>
00335 /// <typeparam name="T">The kind of data describing the data points (generally,
    <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00336     public class DataLabels<T> :  IPlotElement
00337     {
00338 /// <summary>
00339 /// The data points at which the labels will be drawn.
00340 /// </summary>
00341         public IEnumerable<T> Data { get; set; }
00342 
00343         private Func<int, T, IEnumerable<FormattedText» labelFunction;
00344         private Func<int, T, object> label;
00345 
00346 /// <summary>
00347 /// A function used to determine the text of the labels to draw.  The arguments for this function
    should be a <see langword="int"/>
00348 /// representing the index of the data point and a <typeparamref name="T"/> representing the
    coordinates of the data point.  This
00349 /// function should return either an <see cref="IEnumerable{T}"/> of <see cref="FormattedText"/>
    objects, which will be used as-is,
00350 /// or any other kind of object, which will be converted to a <see langword="string"/> to be used in
    the label.
00351 /// </summary>
```

```
00352         public Func<int, T, object> Label
00353         {
00354             get
00355             {
00356                 return label;
00357             }
00358 
00359             set
00360             {
00361                 label = value;
00362 
00363                 if (typeof(IEnumerable<FormattedText>).IsAssignableFrom(value.Method.ReturnType))
00364                 {
00365                     labelFunction = (i, a) => (IEnumerable<FormattedText>)value(i, a);
00366                 }
00367                 else
00368                 {
00369                     labelFunction = (i, a) =>
00370                     {
00371                         object lbl = value(i, a);
00372 
00373                         if (lbl is IEnumerable<FormattedText> frmt)
00374                         {
00375                             return frmt;
00376                         }
00377                         else if (lbl == null)
00378                         {
00379                             return null;
00380                         }
00381                         else
00382                         {
00383                             if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00384                             {
00385                                 int j = Array.IndexOf(FontFamily.StandardFamilies,
    PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00386 
00387                                 return FormattedText.Format(lbl.ToString(),
    (FontFamily.StandardFontFamilies)j, PresentationAttributes.Font.FontSize);
00388                             }
00389                             else
00390                             {
00391                                 return FormattedText.Format(lbl.ToString(),
    PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font,
    PresentationAttributes.Font);
00392                             }
00393                         }
00394                     };
00395                 }
00396             }
00397         }
00398 
00399 /// <summary>
00400 /// A function used to determine the orientation of the labels with respect to the horizontal.  The
    arguments for this function should be a <see langword="int"/>
00401 /// representing the index of the data point and a <typeparamref name="T"/> representing the
    coordinates of the data point.  This
00402 /// function should return a <see langword="double"/> representing the angle with respect to the
    horizontal at which the label
00403 /// should be drawn.
00404 /// </summary>
00405         public Func<int, T, double> Rotation { get; set; } = (i, d) => 0;
00406 
00407 /// <summary>
00408 /// A function used to determine the position of the labels with respect to the data points.  The
    arguments for this function should be a <see langword="int"/>
00409 /// representing the index of the data point and a <typeparamref name="T"/> representing the
    coordinates of the data point.  This
00410 /// function should return a <see cref="Point"/> defining the amount of space between the data point
    and the label, in plot coordinates.
00411 /// </summary>
00412         public Func<int, T, Point> Margin { get; set; } = (i, d) => new Point();
00413 
00414 /// <summary>
00415 /// The baseline for the labels.
00416 /// </summary>
00417         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00418 
00419 /// <summary>
00420 /// The alignment for the labels.
00421 /// </summary>
00422         public TextAnchors Alignment { get; set; } = TextAnchors.Center;
00423 
00424 /// <summary>
00425 /// The coordinate system used to transform the points from data space to plot space.
00426 /// </summary>
00427         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00428         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
```

```
00429
00430 /// <summary>
00431 /// Presentation attributes determining the appearance of the labels.
00432 /// </summary>
00433         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Stroke = null };
00434
00435 /// <summary>
00436 /// A tag to identify the labels in the plot.
00437 /// </summary>
00438         public string Tag { get; set; }
00439
00440 /// <summary>
00441 /// Creates a new <see cref="DataLabels{T}"/> instance.
00442 /// </summary>
00443 /// <param name="data">The data points at which the labels will be drawn.</param>
00444 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00445         public DataLabels(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00446         {
00447             this.Data = data;
00448             this.CoordinateSystem = coordinateSystem;
00449             this.Label = (i, d) =>
00450             {
00451                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00452                 {
00453                     int j = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00454
00455                     return FormattedText.Format(i.ToString(), (FontFamily.StandardFontFamilies)j,
      PresentationAttributes.Font.FontSize);
00456                 }
00457                 else
00458                 {
00459                     return FormattedText.Format(i.ToString(), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00460                 }
00461             };
00462         }
00463
00464 /// <inheritdoc/>
00465         public void Plot(Graphics target)
00466         {
00467             int index = 0;
00468
00469             foreach (T data in Data)
00470             {
00471                 IEnumerable<FormattedText> label = labelFunction(index, data);
00472                 if (label != null)
00473                 {
00474                     double rotation = Rotation(index, data);
00475                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00476                     Point margin = Margin(index, data);
00477
00478                     pt = new Point(pt.X + margin.X, pt.Y + margin.Y);
00479
00480                     target.Save();
00481                     target.Translate(pt);
00482                     target.Rotate(rotation);
00483
00484                     string fillTag = Tag;
00485                     string strokeTag = Tag;
00486
00487
00488                     if (target.UseUniqueTags && !string.IsNullOrEmpty(Tag))
00489                     {
00490                         fillTag = fillTag + "@" + index.ToString();
00491                         strokeTag = strokeTag + "@stroke" + index.ToString();
00492                     }
00493
00494                     if (PresentationAttributes.Fill != null)
00495                     {
00496                         target.FillText(Alignment == TextAnchors.Left ?  0 :  Alignment ==
      TextAnchors.Right ?  -label.Measure().Width :  -label.Measure().Width * 0.5, 0, label,
      PresentationAttributes.Fill, Baseline, fillTag);
00497                     }
00498
00499                     if (PresentationAttributes.Stroke != null)
00500                     {
00501                         target.StrokeText(Alignment == TextAnchors.Left ?  0 :  Alignment ==
      TextAnchors.Right ?  -label.Measure().Width :  -label.Measure().Width * 0.5, 0, label,
      PresentationAttributes.Stroke, Baseline, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      fillTag);
00502                     }
00503
00504                     target.Restore();
```

```
00505                 }
00506                 index++;
00507             }
00508         }
00509     }
00510
00511 /// <summary>
00512 /// A plot element that draws a line passing through a set of points.
00513 /// </summary>
00514 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00515     public class DataLine<T> :  IPlotElement
00516     {
00517 /// <summary>
00518 /// The data points through which the line will pass.
00519 /// </summary>
00520         public IEnumerable<T> Data { get; set; }
00521
00522 /// <summary>
00523 /// If this is <see langword="false"/>, straight line segments are used to join the data points.  If
      this is <see langword="true"/>,
00524 /// a smooth spline passing through all of them is used instead.
00525 /// </summary>
00526         public bool Smooth { get; set; }
00527
00528 /// <summary>
00529 /// The coordinate system used to transform the points from data space to plot space.
00530 /// </summary>
00531         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00532         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00533
00534 /// <summary>
00535 /// Presentation attributes determining the appearance of the line.
00536 /// </summary>
00537         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00538
00539 /// <summary>
00540 /// A tag to identify the labels in the plot.
00541 /// </summary>
00542         public string Tag { get; set; }
00543
00544 /// <summary>
00545 /// Create a new instance of the <see cref="DataLine{T}"/> class.
00546 /// </summary>
00547 /// <param name="data">The data points through which the line will pass.</param>
00548 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00549         public DataLine(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00550         {
00551             this.Data = data;
00552             this.CoordinateSystem = coordinateSystem;
00553         }
00554
00555 /// <inheritdoc/>
00556         public void Plot(Graphics target)
00557         {
00558             GraphicsPath pth = new GraphicsPath();
00559
00560             if (!Smooth)
00561             {
00562                 bool started = false;
00563
00564                 foreach (T data in Data)
00565                 {
00566                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00567
00568                     if (!started)
00569                     {
00570                         pth.MoveTo(pt);
00571                         started = true;
00572                     }
00573                     else
00574                     {
00575                         pth.LineTo(pt);
00576                     }
00577                 }
00578             }
00579             else
00580             {
00581                 List<Point> points = new List<Point>();
00582
00583                 foreach (T data in Data)
00584                 {
00585                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00586                     points.Add(pt);
00587                 }
```

```
00588                 pth.AddSmoothSpline(points.ToArray());
00589             }
00590 
00591             target.StrokePath(pth, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      Tag);
00592         }
00593     }
00594 
00595 /// <summary>
00596 /// A plot element that fills an area between a line passing through some data points and a base line.
00597 /// </summary>
00598 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00599     public class Area<T> :  IPlotElement
00600     {
00601 /// <summary>
00602 /// The data points through which the upper part of the area will pass.
00603 /// </summary>
00604         public IEnumerable<T> Data { get; set; }
00605 
00606 /// <summary>
00607 /// A function returning the baseline for each data point.
00608 /// </summary>
00609         public Func<T, T> GetBaseline { get; set; }
00610 
00611 /// <summary>
00612 /// If this is <see langword="false"/>, straight line segments are used to join the data points.  If
      this is <see langword="true"/>,
00613 /// a smooth spline passing through all of them is used instead.
00614 /// </summary>
00615         public bool Smooth { get; set; }
00616 
00617 
00618 /// <summary>
00619 /// The coordinate system used to transform the points from data space to plot space.
00620 /// </summary>
00621         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00622         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00623 
00624 /// <summary>
00625 /// Presentation attributes determining the appearance of the area.
00626 /// </summary>
00627         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00628 
00629 /// <summary>
00630 /// A tag to identify the area in the plot.
00631 /// </summary>
00632         public string Tag { get; set; }
00633 
00634 /// <summary>
00635 /// Create a new <see cref="Area{T}"/> instance.
00636 /// </summary>
00637 /// <param name="data">The data points through which the upper part of the area will pass.</param>
00638 /// <param name="getBaseline">A function returning the baseline for each data point.</param>
00639 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00640         public Area(IEnumerable<T> data, Func<T, T> getBaseline, ICoordinateSystem<T>
      coordinateSystem)
00641         {
00642             this.Data = data;
00643             this.CoordinateSystem = coordinateSystem;
00644             this.GetBaseline = getBaseline;
00645         }
00646 
00647 /// <inheritdoc/>
00648         public void Plot(Graphics target)
00649         {
00650             GraphicsPath pth = new GraphicsPath();
00651             GraphicsPath pthStroke = new GraphicsPath();
00652 
00653             if (!Smooth)
00654             {
00655                 bool started = false;
00656 
00657                 List<Point> oppositePoints = new List<Point>();
00658 
00659                 foreach (T data in Data)
00660                 {
00661                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00662 
00663                     oppositePoints.Add(CoordinateSystem.ToPlotCoordinates(GetBaseline(data)));
00664 
00665                     if (!started)
00666                     {
00667                         pth.MoveTo(pt);
```

```
00668                           pthStroke.MoveTo(pt);
00669                           started = true;
00670                       }
00671                       else
00672                       {
00673                           pth.LineTo(pt);
00674                           pthStroke.LineTo(pt);
00675                       }
00676                   }
00677
00678                   for (int i = oppositePoints.Count - 1; i >= 0; i--)
00679                   {
00680                       pth.LineTo(oppositePoints[i]);
00681                   }
00682
00683                   pth.Close();
00684               }
00685               else
00686               {
00687                   List<Point> points = new List<Point>();
00688                   List<Point> oppositePoints = new List<Point>();
00689
00690                   foreach (T data in Data)
00691                   {
00692                       Point pt = CoordinateSystem.ToPlotCoordinates(data);
00693                       points.Add(pt);
00694                       oppositePoints.Add(CoordinateSystem.ToPlotCoordinates(GetBaseline(data)));
00695                   }
00696                   pth.AddSmoothSpline(points.ToArray());
00697                   pthStroke.AddSmoothSpline(points.ToArray());
00698                   oppositePoints.Reverse();
00699                   pth.AddSmoothSpline(oppositePoints.ToArray());
00700                   pth.Close();
00701               }
00702
00703               string tag = Tag;
00704               string strokeTag = tag;
00705
00706               if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00707               {
00708                   strokeTag += "@stroke";
00709               }
00710
00711               if (PresentationAttributes.Fill != null)
00712               {
00713                   target.FillPath(pth, PresentationAttributes.Fill, tag);
00714               }
00715
00716               if (PresentationAttributes.Stroke != null)
00717               {
00718                   target.StrokePath(pthStroke, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00719               }
00720           }
00721       }
00722 }
```

8.26 Function2D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using MathNet.Numerics.Distributions;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023
```

```
00024 namespace VectSharp.Plots
00025 {
00026 /// <summary>
00027 /// Represents a function of two variables that has been sampled in some points.
00028 /// </summary>
00029     public class Function2DGrid
00030     {
00031 /// <summary>
00032 /// Describes the arrangement of the points that have been sampled.
00033 /// </summary>
00034         public enum GridType
00035         {
00036 /// <summary>
00037 /// Points have been sampled along a rectangular grid.
00038 /// </summary>
00039             Rectangular,
00040
00041 /// <summary>
00042 /// Points have been sampled along a grid composed of hexagons with a diagonal
00043 /// parallel to the horizontal.
00044 /// </summary>
00045             HexagonHorizontal,
00046
00047 /// <summary>
00048 /// Points have been sampled along a grid composed of hexagons with a diagonal
00049 /// parallel to the vertical.
00050 /// </summary>
00051             HexagonVertical,
00052
00053 /// <summary>
00054 /// Points have been sampled without any particular criterion.
00055 /// </summary>
00056             Irregular
00057         }
00058
00059 /// <summary>
00060 /// The points where the function has been sampled.
00061 /// </summary>
00062         public IReadOnlyList<IReadOnlyList<double> DataPoints { get; }
00063
00064 /// <summary>
00065 /// The minimum X value that has been sampled.
00066 /// </summary>
00067         public double MinX { get; }
00068
00069 /// <summary>
00070 /// The maximum X value that has been sampled.
00071 /// </summary>
00072         public double MaxX { get; }
00073
00074 /// <summary>
00075 /// The minimum Y value that has been sampled.
00076 /// </summary>
00077         public double MinY { get; }
00078
00079 /// <summary>
00080 /// The maximum Y value that has been sampled.
00081 /// </summary>
00082         public double MaxY { get; }
00083
00084 /// <summary>
00085 /// The minimum value that has been obtained for the function.
00086 /// </summary>
00087         public double MinZ { get; }
00088
00089 /// <summary>
00090 /// The maximum value that has been obtained for the function.
00091 /// </summary>
00092         public double MaxZ { get; }
00093
00094 /// <summary>
00095 /// If the function has been sampled along a regular grid, the number of steps
00096 /// between <see cref="MinX"/> and <see cref="MaxX"/> on the X axis.
00097 /// </summary>
00098         public int StepsX { get; private set; }
00099
00100 /// <summary>
00101 /// If the function has been sampled along a regular grid, the number of steps
00102 /// between <see cref="MinY"/> and <see cref="MaxY"/> on the X axis.
00103 /// </summary>
00104         public int StepsY { get; private set; }
00105
00106 /// <summary>
00107 /// The type of grid.
00108 /// </summary>
00109         public GridType Type { get; private set; }
00110
```

```
00111 /// <summary>
00112 /// Create a new <see cref="Function2DGrid"/> from a list of sampled values.
00113 /// </summary>
00114 /// <param name="dataPoints">The sampled values.  Each element of this <see cref="IReadOnlyList{T}"/>
      should
00115 /// be a collection of three values:  the X coordinate of the sampled point, the Y coordinate of the
      sampled
00116 /// point, and the value of the function at that point.</param>
00117         public Function2DGrid(IReadOnlyList<IReadOnlyList<double» dataPoints)
00118         {
00119             this.Type = GridType.Irregular;
00120             this.DataPoints = dataPoints.ToArray();
00121 
00122             MinX = double.MaxValue;
00123             MaxX = double.MinValue;
00124             MinY = double.MaxValue;
00125             MaxY = double.MinValue;
00126             MinZ = double.MaxValue;
00127             MaxZ = double.MinValue;
00128 
00129             this.StepsX = 0;
00130             this.StepsY = 0;
00131 
00132             for (int i = 0; i < dataPoints.Count; i++)
00133             {
00134                 MinX = Math.Min(MinX, dataPoints[i][0]);
00135                 MinY = Math.Min(MinY, dataPoints[i][1]);
00136                 MinZ = Math.Min(MinZ, dataPoints[i][2]);
00137 
00138                 MaxX = Math.Max(MaxX, dataPoints[i][0]);
00139                 MaxY = Math.Max(MaxY, dataPoints[i][1]);
00140                 MaxZ = Math.Max(MaxZ, dataPoints[i][2]);
00141             }
00142         }
00143 
00144 /// <summary>
00145 /// Create a new <see cref="Function2DGrid"/> by sampling the provided function.
00146 /// </summary>
00147 /// <param name="function">The function to sample.</param>
00148 /// <param name="minX">The minimum X value at which the function should be sampled.</param>
00149 /// <param name="minY">The minimum Y value at which the function should be sampled.</param>
00150 /// <param name="maxX">The maximum X value at which the function should be sampled.</param>
00151 /// <param name="maxY">The maximum Y value at which the function should be sampled.</param>
00152 /// <param name="stepsX">If <paramref name="type"/> is not <see cref="GridType.Irregular"/>,
00153 /// the number of steps between <paramref name="minX"/> and <paramref name="maxX"/> on the X axis.
00154 /// Otherwise, the number of sampled points is determined by multiplying <paramref name="stepsX"/> and
00155 /// <paramref name="stepsY"/> together.</param>
00156 /// <param name="stepsY">If <paramref name="type"/> is not <see cref="GridType.Irregular"/>,
00157 /// the number of steps between <paramref name="minY"/> and <paramref name="maxY"/> on the Y axis.
00158 /// Otherwise, the number of sampled points is determined by multiplying <paramref name="stepsX"/> and
00159 /// <paramref name="stepsY"/> together.</param>
00160 /// <param name="type">The strategy used to select points to sample.  If this is <see
      cref="GridType.Irregular"/>,
00161 /// uniformly distributed random points between (<paramref name="minX"/>, <paramref name="minY"/>) and
00162 /// (<paramref name="maxX"/>, <paramref name="maxY"/>) are sampled.</param>
00163         public Function2DGrid(Func<double[], double> function, double minX, double minY, double maxX,
      double maxY, int stepsX, int stepsY, GridType type)
00164         {
00165             MinX = minX;
00166             MaxX = maxX;
00167             MinY = minY;
00168             MaxY = maxY;
00169             StepsX = stepsX;
00170             StepsY = stepsY;
00171             Type = type;
00172             MinZ = double.MaxValue;
00173             MaxZ = double.MinValue;
00174 
00175             if (type == GridType.HexagonHorizontal)
00176             {
00177                 List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00178 
00179                 for (int y = 0; y <= stepsY; y++)
00180                 {
00181                     for (int x = 0; x <= stepsX; x++)
00182                     {
00183                         if (x % 2 == 0 || y < stepsY)
00184                         {
00185 
00186                             double realY;
00187 
00188                             if (x % 2 == 0)
00189                             {
00190                                 realY = minY + (maxY - minY) / stepsY * y;
00191                             }
00192                             else
00193                             {
```

```
00194                                 realY = minY + (maxY - minY) / stepsY * (y + 0.5);
00195                             }
00196
00197                             double realX = minX + (maxX - minX) / stepsX * x;
00198                             double z = function(new double[] { realX, realY });
00199
00200                             if (!double.IsNaN(z))
00201                             {
00202                                 MinZ = Math.Min(MinZ, z);
00203                                 MaxZ = Math.Max(MaxZ, z);
00204
00205                                 dataPoints.Add(new double[] { realX, realY, z });
00206                             }
00207                             else
00208                             {
00209                                 Type = GridType.Irregular;
00210                             }
00211                         }
00212                     }
00213                 }
00214
00215                 this.DataPoints = dataPoints;
00216             }
00217             else if (type == GridType.HexagonVertical)
00218             {
00219                 List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00220
00221                 for (int y = 0; y <= stepsY; y++)
00222                 {
00223                     double realY = minY + (maxY - minY) / stepsY * y;
00224
00225                     for (int x = 0; x <= stepsX; x++)
00226                     {
00227                         if (y % 2 == 0 || x < stepsX)
00228                         {
00229
00230                             double realX;
00231
00232                             if (y % 2 == 0)
00233                             {
00234                                 realX = minX + (maxX - minX) / stepsX * x;
00235                             }
00236                             else
00237                             {
00238                                 realX = minX + (maxX - minX) / stepsX * (x + 0.5);
00239                             }
00240
00241                             double z = function(new double[] { realX, realY });
00242
00243                             if (!double.IsNaN(z))
00244                             {
00245                                 MinZ = Math.Min(MinZ, z);
00246                                 MaxZ = Math.Max(MaxZ, z);
00247
00248                                 dataPoints.Add(new double[] { realX, realY, z });
00249                             }
00250                             else
00251                             {
00252                                 Type = GridType.Irregular;
00253                             }
00254                         }
00255                     }
00256                 }
00257
00258                 this.DataPoints = dataPoints;
00259             }
00260             else if (type == GridType.Rectangular)
00261             {
00262                 List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00263
00264                 for (int y = 0; y <= stepsY; y++)
00265                 {
00266                     double realY = minY + (maxY - minY) / stepsY * y;
00267                     for (int x = 0; x <= stepsX; x++)
00268                     {
00269                         double realX = minX + (maxX - minX) / stepsX * x;
00270                         double z = function(new double[] { realX, realY });
00271
00272                         if (!double.IsNaN(z))
00273                         {
00274                             MinZ = Math.Min(MinZ, z);
00275                             MaxZ = Math.Max(MaxZ, z);
00276
00277                             dataPoints.Add(new double[] { realX, realY, z });
00278                         }
00279                         else
00280                         {
```

```
00281                               Type = GridType.Irregular;
00282                           }
00283                       }
00284                   }
00285
00286                   this.DataPoints = dataPoints;
00287               }
00288               else //type == GridType.Irregular
00289               {
00290                   double[] xs = ContinuousUniform.Samples(minX, maxX).Take((stepsX + 1) * (stepsY +
      1)).ToArray();
00291                   double[] ys = ContinuousUniform.Samples(minY, maxY).Take((stepsX + 1) * (stepsY +
      1)).ToArray();
00292
00293                   List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00294
00295                   for (int i = 0; i < (stepsX + 1) * (stepsY + 1); i++)
00296                   {
00297                       double z = function(new double[] { xs[i], ys[i] });
00298
00299                       if (!double.IsNaN(z))
00300                       {
00301                           MinZ = Math.Min(MinZ, z);
00302                           MaxZ = Math.Max(MaxZ, z);
00303
00304                           dataPoints.Add(new double[] { xs[i], ys[i], z });
00305                       }
00306                   }
00307
00308                   this.DataPoints = dataPoints;
00309               }
00310           }
00311
00312 /// <summary>
00313 /// Converts a hexagonal grid into a rectangular grid.
00314 /// </summary>
00315 /// <returns>If <see cref="Type"/> is <see cref="GridType.Rectangular"/>, this method returns the
      current instance.  <br/>
00316 /// If <see cref="Type"/> is <see cref="GridType.HexagonHorizontal"/> or <see
      cref="GridType.HexagonVertical"/>, a
00317 /// new <see cref="Function2DGrid"/> with <see cref="Type"/> equal to <see
      cref="GridType.Rectangular"/> is returned.
00318 /// The sampled points in this grid are obtained by performing a bilinear interpolation on the sampled
      points from
00319 /// this grid.  The returned grid will always be "denser" than the current instance.<br/>
00320 /// If <see cref="Type"/> is <see cref="GridType.Irregular"/>, an <see
      cref="InvalidOperationException"/> is thrown.</returns>
00321 /// <exception cref="InvalidOperationException">Thrown if <see cref="Type"/> is <see
      cref="GridType.Irregular"/>.</exception>
00322           public Function2DGrid ToRectangular()
00323           {
00324               if (this.Type == GridType.Rectangular)
00325               {
00326                   return this;
00327               }
00328               else if (this.Type == GridType.HexagonHorizontal)
00329               {
00330                   IReadOnlyList<double>[] data = new IReadOnlyList<double>[(2 * this.StepsY + 1) *
      (this.StepsX + 1)];
00331
00332                   int newStepsX = this.StepsX;
00333                   int newStepsY = this.StepsY * 2;
00334
00335                   for (int i = 0; i < this.DataPoints.Count; i++)
00336                   {
00337                       int x = (int)Math.Round((this.DataPoints[i][0] - this.MinX) / (this.MaxX -
      this.MinX) * newStepsX);
00338                       int y = (int)Math.Round((this.DataPoints[i][1] - this.MinY) / (this.MaxY -
      this.MinY) * newStepsY);
00339
00340                       data[y * (newStepsX + 1) + x] = this.DataPoints[i];
00341                   }
00342
00343                   for (int y = 0; y <= newStepsY; y++)
00344                   {
00345                       for (int x = 0; x <= newStepsX; x++)
00346                       {
00347                           if (data[y * (newStepsX + 1) + x] == null)
00348                           {
00349                               double xVal = this.MinX + (this.MaxX - this.MinX) / newStepsX * x;
00350                               double yVal = this.MinY + (this.MaxY - this.MinY) / newStepsY * y;
00351
00352                               double zVal;
00353
00354                               if (x > 0)
00355                               {
00356                                   if (y > 0)
```

```
00357                                 {
00358                                     if (x < newStepsX)
00359                                     {
00360                                         if (y < newStepsY)
00361                                         {
00362                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2] + data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.25;
00363                                         }
00364                                         else
00365                                         {
00366                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.5;
00367                                         }
00368                                     }
00369                                     else
00370                                     {
00371                                         if (y < newStepsY)
00372                                         {
00373                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00374                                         }
00375                                         else
00376                                         {
00377                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y - 1) * (newStepsX + 1) + x][2]) * 0.5;
00378                                         }
00379                                     }
00380                                 }
00381                                 else
00382                                 {
00383                                     if (x < newStepsX)
00384                                     {
00385                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.5;
00386                                     }
00387                                     else
00388                                     {
00389                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00390                                     }
00391                                 }
00392                             }
00393                             else
00394                             {
00395                                 if (y > 0)
00396                                 {
00397                                     if (y < newStepsY)
00398                                     {
00399                                         zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00400                                     }
00401                                     else
00402                                     {
00403                                         zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y - 1) * (newStepsX + 1) + x][2]) * 0.5;
00404                                     }
00405                                 }
00406                                 else
00407                                 {
00408                                     zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00409                                 }
00410                             }
00411 
00412                             data[y * (newStepsX + 1) + x] = new double[] { xVal, yVal, zVal };
00413                         }
00414                     }
00415                 }
00416 
00417                 return new Function2DGrid(data) { Type = GridType.Rectangular, StepsX = newStepsX, StepsY = newStepsY };
00418             }
00419             else if (this.Type == GridType.HexagonVertical)
00420             {
00421                 IReadOnlyList<double>[] data = new IReadOnlyList<double>[(this.StepsY + 1) * (2 * this.StepsX + 1)];
00422 
00423                 int newStepsX = this.StepsX * 2;
00424                 int newStepsY = this.StepsY;
00425 
00426                 for (int i = 0; i < this.DataPoints.Count; i++)
00427                 {
00428                     int x = (int)Math.Round((this.DataPoints[i][0] - this.MinX) / (this.MaxX - this.MinX) * newStepsX);
00429                     int y = (int)Math.Round((this.DataPoints[i][1] - this.MinY) / (this.MaxY - this.MinY) * newStepsY);
```

```
00430
00431                     data[y * (newStepsX + 1) + x] = this.DataPoints[i];
00432                 }
00433
00434                 for (int y = 0; y <= newStepsY; y++)
00435                 {
00436                     for (int x = 0; x <= newStepsX; x++)
00437                     {
00438                         if (data[y * (newStepsX + 1) + x] == null)
00439                         {
00440                             double xVal = this.MinX + (this.MaxX - this.MinX) / newStepsX * x;
00441                             double yVal = this.MinY + (this.MaxY - this.MinY) / newStepsY * y;
00442
00443                             double zVal;
00444
00445                             if (x > 0)
00446                             {
00447                                 if (y > 0)
00448                                 {
00449                                     if (x < newStepsX)
00450                                     {
00451                                         if (y < newStepsY)
00452                                         {
00453                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y +
      1) * (newStepsX + 1) + x][2] + data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x
      + 1)][2]) * 0.25;
00454                                         }
00455                                         else
00456                                         {
00457                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y *
      (newStepsX + 1) + (x + 1)][2]) * 0.5;
00458                                         }
00459                                     }
00460                                     else
00461                                     {
00462                                         if (y < newStepsY)
00463                                         {
00464                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y +
      1) * (newStepsX + 1) + x][2]) * 0.5;
00465                                         }
00466                                         else
00467                                         {
00468                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y -
      1) * (newStepsX + 1) + x][2]) * 0.5;
00469                                         }
00470                                     }
00471                                 }
00472                                 else
00473                                 {
00474                                     if (x < newStepsX)
00475                                     {
00476                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y *
      (newStepsX + 1) + (x + 1)][2]) * 0.5;
00477                                     }
00478                                     else
00479                                     {
00480                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y + 1)
      * (newStepsX + 1) + x][2]) * 0.5;
00481                                     }
00482                                 }
00483                             }
00484                             else
00485                             {
00486                                 if (y > 0)
00487                                 {
00488                                     if (y < newStepsY)
00489                                     {
00490                                         zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1)
      * (newStepsX + 1) + x][2]) * 0.5;
00491                                     }
00492                                     else
00493                                     {
00494                                         zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y - 1)
      * (newStepsX + 1) + x][2]) * 0.5;
00495                                     }
00496                                 }
00497                                 else
00498                                 {
00499                                     zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y + 1) *
      (newStepsX + 1) + x][2]) * 0.5;
00500                                 }
00501                             }
00502
00503                             data[y * (newStepsX + 1) + x] = new double[] { xVal, yVal, zVal };
00504                         }
00505                     }
00506                 }
```

```
00507
00508                 return new Function2DGrid(data) { Type = GridType.Rectangular, StepsX = newStepsX,
      StepsY = newStepsY };
00509             }
00510             else
00511             {
00512                 throw new InvalidOperationException("Cannot convert an irregular grid into a
      rectangular grid!");
00513             }
00514         }
00515     }
00516
00517 /// <summary>
00518 /// A plot element that plots a function of two variables.
00519 /// </summary>
00520     public class Function2D :  IPlotElement
00521     {
00522 /// <summary>
00523 /// Describes the kind of plots that can be produced.
00524 /// </summary>
00525         public enum PlotType
00526         {
00527 /// <summary>
00528 /// A symbol is drawn at each sampled point, whose
00529 /// colour depends on the value of the function at that point.
00530 /// </summary>
00531             SampledPoints,
00532
00533 /// <summary>
00534 /// The plot area is tessellated with cells whose colour
00535 /// depends on the value of the function at a point within the cell.
00536 /// </summary>
00537             Tessellation,
00538
00539 /// <summary>
00540 /// A rasterised tessellation is created and then stretched and interpolated
00541 /// to fill the plot area.
00542 /// </summary>
00543             Raster
00544         }
00545
00546 /// <summary>
00547 /// The symbol to draw at the sampled points.
00548 /// </summary>
00549         public IDataPointElement SampledPointElement { get; set; } = new PathDataPointElement();
00550
00551 /// <summary>
00552 /// Resolution on the X axis for the rasterised tesselation.
00553 /// </summary>
00554         public int RasterResolutionX { get; set; } = 512;
00555
00556 /// <summary>
00557 /// Resolution on the Y axis for the rasterised tesselation.
00558 /// </summary>
00559         public int RasterResolutionY { get; set; } = 512;
00560
00561 /// <summary>
00562 /// The function to plot.
00563 /// </summary>
00564         public Function2DGrid Function { get; set; }
00565
00566 /// <summary>
00567 /// The kind of plot that is produced.
00568 /// </summary>
00569         public PlotType Type { get; set; } = PlotType.SampledPoints;
00570
00571 /// <summary>
00572 /// A function associating sampled function values to a <see cref="Colour"/>.  You should
00573 /// set this to a function accepting a single <see langword="double"/> argument ranging between
00574 /// 0 and 1, and returning the corresponding colour.  The default value returns black 0 and white for
      1.
00575 /// </summary>
00576         public Func<double, Colour> Colouring { get; set; } = x => { x = Math.Max(0, Math.Min(1, x));
      return Colour.FromRgb(x, x, x); };
00577
00578 /// <summary>
00579 /// A tag to identify the function in the plot.
00580 /// </summary>
00581         public string Tag { get; set; }
00582
00583 /// <summary>
00584 /// The coordinate system used to transform the points from data space to plot space.
00585 /// </summary>
00586         public IContinuousInvertibleCoordinateSystem CoordinateSystem { get; set; }
00587         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00588
00589 /// <summary>
```

```
00590 /// Create a new <see cref="Function2D"/> instance.
00591 /// </summary>
00592 /// <param name="function">The function to plot.</param>
00593 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00594         public Function2D(Function2DGrid function, IContinuousInvertibleCoordinateSystem
      coordinateSystem)
00595         {
00596             this.Function = function;
00597             this.CoordinateSystem = coordinateSystem;
00598         }
00599 
00600 /// <inheritdoc/>
00601         public unsafe void Plot(Graphics target)
00602         {
00603             if (Type == PlotType.SampledPoints)
00604             {
00605                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
      Function.MaxY });
00606                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
      Function.MinY });
00607 
00608                 double size = Math.Min(Math.Abs(topLeft.X - bottomRight.X), Math.Abs(topLeft.Y -
      bottomRight.Y)) * 0.0075;
00609 
00610                 if (Function.Type != Function2DGrid.GridType.Irregular)
00611                 {
00612                     size = Math.Min(Math.Abs(topLeft.X - bottomRight.X) / Function.StepsX,
      Math.Abs(topLeft.Y - bottomRight.Y) / Function.StepsY) * 0.4;
00613                 }
00614 
00615                 for (int i = 0; i < Function.DataPoints.Count; i++)
00616                 {
00617                     Point pt = CoordinateSystem.ToPlotCoordinates(Function.DataPoints[i]);
00618                     Colour col = Colouring((Function.DataPoints[i][2] - Function.MinZ) /
      (Function.MaxZ - Function.MinZ));
00619 
00620                     string tag = Tag;
00621                     if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00622                     {
00623                         tag += "@" + i.ToString();
00624                     }
00625 
00626                     target.Save();
00627                     target.Translate(pt);
00628                     target.Scale(size, size);
00629 
00630                     SampledPointElement.Plot(target, new PlotElementPresentationAttributes() { Fill =
      col, Stroke = null }, tag);
00631 
00632                     target.Restore();
00633                 }
00634             }
00635             else if (Type == PlotType.Tessellation || (Type == PlotType.Raster && Function.Type ==
      Function2DGrid.GridType.Irregular))
00636             {
00637                 double width = (Function.MaxX - Function.MinX) / Function.StepsX;
00638                 double height = (Function.MaxY - Function.MinY) / Function.StepsY;
00639 
00640                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
      Function.MaxY });
00641                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
      Function.MaxY });
00642                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
      Function.MinY });
00643                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
      Function.MinY });
00644 
00645 
00646                 Graphics strokes = new Graphics();
00647                 Graphics fills = new Graphics();
00648 
00649                 bool anyTransparent = false;
00650 
00651                 if (Function.Type != Function2DGrid.GridType.Irregular)
00652                 {
00653                     for (int i = 0; i < Function.DataPoints.Count; i++)
00654                     {
00655                         Colour col = Colouring((Function.DataPoints[i][2] - Function.MinZ) /
      (Function.MaxZ - Function.MinZ));
00656 
00657                         if (col.A != 1)
00658                         {
00659                             anyTransparent = true;
00660                         }
00661 
00662                         if (Function.Type == Function2DGrid.GridType.Rectangular)
```

```
00663                         {
00664                             Point p1 = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] - width * 0.5, Function.DataPoints[i][1] - height * 0.5 });
00665                             Point p2 = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] - width * 0.5, Function.DataPoints[i][1] + height * 0.5 });
00666                             Point p3 = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] + width * 0.5, Function.DataPoints[i][1] + height * 0.5 });
00667                             Point p4 = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] + width * 0.5, Function.DataPoints[i][1] - height * 0.5 });
00668
00669                             GraphicsPath pth = new
      GraphicsPath().MoveTo(p1).LineTo(p2).LineTo(p3).LineTo(p4).Close();
00670
00671                             string tag = Tag;
00672                             string strokeTag = Tag;
00673                             if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00674                             {
00675                                 tag += "@" + i.ToString();
00676                                 strokeTag += "@stroke" + i.ToString();
00677                             }
00678
00679                             strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00680                             fills.FillPath(pth, col, tag);
00681                         }
00682                         else if (Function.Type == Function2DGrid.GridType.HexagonHorizontal)
00683                         {
00684                             double rY = height * 0.57735026919;
00685                             double rX = width * 2 / 3;
00686
00687                             Point[] points = new Point[6];
00688
00689                             for (int j = 0; j < 6; j++)
00690                             {
00691                                 points[j] = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] + rX * Math.Cos(Math.PI / 3 * j), Function.DataPoints[i][1] + rY *
      Math.Sin(Math.PI / 3 * j) });
00692                             }
00693
00694                             GraphicsPath pth = new
      GraphicsPath().MoveTo(points[0]).LineTo(points[1]).LineTo(points[2]).LineTo(points[3]).LineTo(points[4]).LineTo(points[5
00695
00696                             string tag = Tag;
00697                             string strokeTag = Tag;
00698                             if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00699                             {
00700                                 tag += "@" + i.ToString();
00701                                 strokeTag += "@stroke" + i.ToString();
00702                             }
00703
00704                             strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00705                             fills.FillPath(pth, col, tag);
00706                         }
00707                         else if (Function.Type == Function2DGrid.GridType.HexagonVertical)
00708                         {
00709                             double rX = width * 0.57735026919;
00710                             double rY = height * 2 / 3;
00711
00712                             Point[] points = new Point[6];
00713
00714                             for (int j = 0; j < 6; j++)
00715                             {
00716                                 points[j] = CoordinateSystem.ToPlotCoordinates(new double[] {
      Function.DataPoints[i][0] + rX * Math.Sin(Math.PI / 3 * j), Function.DataPoints[i][1] + rY *
      Math.Cos(Math.PI / 3 * j) });
00717                             }
00718
00719                             GraphicsPath pth = new
      GraphicsPath().MoveTo(points[0]).LineTo(points[1]).LineTo(points[2]).LineTo(points[3]).LineTo(points[4]).LineTo(points[5
00720
00721                             string tag = Tag;
00722                             string strokeTag = Tag;
00723                             if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00724                             {
00725                                 tag += "@" + i.ToString();
00726                                 strokeTag += "@stroke" + i.ToString();
00727                             }
00728
00729                             strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00730                             fills.FillPath(pth, col, tag);
00731                         }
00732                     }
00733                 }
00734                 else
00735                 {
00736                     double[][] plotCoordinates = new double[Function.DataPoints.Count][];
00737
00738                     double minX = double.MaxValue;
```

```
00739                     double minY = double.MaxValue;
00740                     double maxX = double.MinValue;
00741                     double maxY = double.MinValue;
00742 
00743                     for (int i = 0; i < Function.DataPoints.Count; i++)
00744                     {
00745                         Point pt = CoordinateSystem.ToPlotCoordinates(Function.DataPoints[i]);
00746                         minX = Math.Min(minX, pt.X);
00747                         minY = Math.Min(minY, pt.Y);
00748                         maxX = Math.Max(maxX, pt.X);
00749                         maxY = Math.Max(maxY, pt.Y);
00750                         plotCoordinates[i] = new double[] { pt.X, pt.Y };
00751                     }
00752 
00753                     List<double[][]> cells = Voronoi.Voronoi.GetVoronoiCells(plotCoordinates, minX,
      minY, maxX, maxY);
00754 
00755                     for (int j = 0; j < cells.Count; j++)
00756                     {
00757                         Colour col = Colouring((Function.DataPoints[j][2] - Function.MinZ) /
      (Function.MaxZ - Function.MinZ));
00758 
00759                         if (col.A != 1)
00760                         {
00761                             anyTransparent = true;
00762                         }
00763 
00764                         GraphicsPath path = new GraphicsPath();
00765 
00766                         for (int k = 0; k < cells[j].Length; k++)
00767                         {
00768                             path.LineTo(cells[j][k][0], cells[j][k][1]);
00769                         }
00770 
00771                         path.Close();
00772 
00773                         string tag = Tag;
00774                         string strokeTag = Tag;
00775                         if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00776                         {
00777                             tag += "@" + j.ToString();
00778                             strokeTag += "@stroke" + j.ToString();
00779                         }
00780 
00781                         strokes.StrokePath(path, col, 0.5, tag:  strokeTag);
00782                         fills.FillPath(path, col, tag);
00783 
00784                     }
00785                 }
00786 
00787                 target.Save();
00788                 target.SetClippingPath(new
      GraphicsPath().MoveTo(topLeft).LineTo(topRight).LineTo(bottomRight).LineTo(bottomLeft).Close());
00789 
00790                 if (Type != PlotType.Raster)
00791                 {
00792                     if (!anyTransparent)
00793                     {
00794                         target.DrawGraphics(0, 0, strokes);
00795                     }
00796 
00797                     target.DrawGraphics(0, 0, fills);
00798                 }
00799                 else
00800                 {
00801                     double x1 = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00802                     double x0 = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
00803 
00804                     double y1 = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00805                     double y0 = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00806 
00807                     double scaleX = RasterResolutionX / (x1 - x0);
00808                     double scaleY = RasterResolutionY / (y1 - y0);
00809 
00810                     Graphics toBeRasterised = new Graphics();
00811                     toBeRasterised.Scale(scaleX, scaleY);
00812 
00813                     if (!anyTransparent)
00814                     {
00815                         toBeRasterised.DrawGraphics(0, 0, strokes);
00816                     }
00817 
00818                     toBeRasterised.DrawGraphics(0, 0, fills);
```

```
00819
00820                     if (toBeRasterised.TryRasterise(new Rectangle(x0 * scaleX, y0 * scaleY, (x1 - x0)
    * scaleX, (y1 - y0) * scaleY), 1, true, out RasterImage image))
00821                     {
00822                         target.DrawRasterImage(x0, y0, x1 - x0, y1 - y0, image, Tag);
00823                     }
00824                 }
00825                 target.Restore();
00826
00827             }
00828             else if (Type == PlotType.Raster)
00829             {
00830                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
    Function.MaxY });
00831                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
    Function.MaxY });
00832                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
    Function.MinY });
00833                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
    Function.MinY });
00834
00835                 double width = Math.Abs(topLeft.X - bottomRight.X);
00836                 double height = Math.Abs(topLeft.Y - bottomRight.Y);
00837
00838                 RasterImage image = null;
00839
00840                 Function2DGrid function = Function;
00841
00842                 if (Function.Type != Function2DGrid.GridType.Rectangular)
00843                 {
00844                     function = Function.ToRectangular();
00845                 }
00846
00847
00848                 double[,] imageData = new double[function.StepsX + 1, function.StepsY + 1];
00849
00850                 int sampleWidth = function.StepsX + 1;
00851                 int sampleHeight = function.StepsY + 1;
00852
00853                 for (int i = 0; i < function.DataPoints.Count; i++)
00854                 {
00855                     int x = (int)Math.Round((function.DataPoints[i][0] - function.MinX) /
    (function.MaxX - function.MinX) * function.StepsX);
00856                     int y = (int)Math.Round((function.DataPoints[i][1] - function.MinY) /
    (function.MaxY - function.MinY) * function.StepsY);
00857
00858                     imageData[x, y] = (function.DataPoints[i][2] - function.MinZ) / (function.MaxZ -
    function.MinZ);
00859                 }
00860
00861                 IntPtr imageAddr = Marshal.AllocHGlobal(RasterResolutionX * RasterResolutionY * 4);
00862                 DisposableIntPtr disp = new DisposableIntPtr(imageAddr);
00863
00864                 unsafe
00865                 {
00866                     byte* imageBytes = (byte*)imageAddr;
00867
00868                     for (int y = 0; y < RasterResolutionY; y++)
00869                     {
00870                         for (int x = 0; x < RasterResolutionX; x++)
00871                         {
00872                             double realX = (double)x / (RasterResolutionX - 1);
00873                             double realY = 1 - (double)y / (RasterResolutionY - 1);
00874
00875                             double intensity00 = imageData[(int)Math.Floor(realX * (sampleWidth - 1)),
    (int)Math.Floor(realY * (sampleHeight - 1))];
00876                             double intensity01 = imageData[(int)Math.Floor(realX * (sampleWidth - 1)),
    Math.Min(sampleHeight - 1, (int)Math.Floor(realY * (sampleHeight - 1)) + 1)];
00877                             double intensity10 = imageData[Math.Min(sampleWidth - 1,
    (int)Math.Floor(realX * (sampleWidth - 1)) + 1), (int)Math.Floor(realY * (sampleHeight - 1))];
00878                             double intensity11 = imageData[Math.Min(sampleWidth - 1,
    (int)Math.Floor(realX * (sampleWidth - 1)) + 1), Math.Min(sampleHeight - 1, (int)Math.Floor(realY *
    (sampleHeight - 1)) + 1)];
00879
00880                             double fracX = realX * (sampleWidth - 1) - Math.Floor(realX * (sampleWidth
    - 1));
00881                             double fracY = realY * (sampleHeight - 1) - Math.Floor(realY *
    (sampleHeight - 1));
00882
00883                             double intensityX0 = intensity00 * (1 - fracX) + intensity10 * fracX;
00884                             double intensityX1 = intensity01 * (1 - fracX) + intensity11 * fracX;
00885                             double intensity = intensityX0 * (1 - fracY) + intensityX1 * fracY;
00886
00887                             Colour col = Colouring(intensity);
00888
00889                             imageBytes[(y * RasterResolutionX + x) * 4] = (byte)(col.R * 255);
00890                             imageBytes[(y * RasterResolutionX + x) * 4 + 1] = (byte)(col.G * 255);
```

```
00891                               imageBytes[(y * RasterResolutionX + x) * 4 + 2] = (byte)(col.B * 255);
00892                               imageBytes[(y * RasterResolutionX + x) * 4 + 3] = (byte)(col.A * 255);
00893                           }
00894                       }
00895                   }
00896
00897                   image = new RasterImage(ref disp, RasterResolutionX, RasterResolutionY, true, true);
00898
00899
00900                   if (image != null)
00901                   {
00902                       target.Save();
00903                       target.SetClippingPath(new
      GraphicsPath().MoveTo(topLeft).LineTo(topRight).LineTo(bottomRight).LineTo(bottomLeft).Close());
00904
00905                       target.DrawRasterImage(topLeft, new Size(width, height), image, Tag);
00906
00907                       target.Restore();
00908                   }
00909               }
00910           }
00911       }
00912 }
00913
```

8.27 Pie.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// A plot element that draws a pie or a doughnut.
00026 /// </summary>
00027     public class Pie :  IPlotElement
00028     {
00029 /// <summary>
00030 /// The data in the pie chart.  The values do not need to be normalised.
00031 /// </summary>
00032         public IReadOnlyList<double> Data { get; set; }
00033
00034 /// <summary>
00035 /// The centre of the pie/doughnut, in data coordinates.
00036 /// </summary>
00037         public IReadOnlyList<double> Centre { get; set; }
00038
00039 /// <summary>
00040 /// The outer radius of the pie/doughnut, in data coordinates.
00041 /// </summary>
00042         public IReadOnlyList<double> OuterRadius { get; set; }
00043
00044 /// <summary>
00045 /// The inner radius of the doughnut, in data coordinates.  Set to [0, 0] for a pie chart.
00046 /// </summary>
00047         public IReadOnlyList<double> InnerRadius { get; set; }
00048
00049 /// <summary>
00050 /// The initial angle starting from which the pie slices are drawn.
00051 /// </summary>
00052         public double StartAngle { get; set; } = 0;
00053
00054 /// <summary>
00055 /// Determines whether the slices are drawn in clockwise or anti-clockwise fashion.
00056 /// </summary>
```

```
00057         public bool Clockwise { get; set; } = false;
00058
00059 /// <summary>
00060 /// The coordinate system used to transform the points from data space to plot space.
00061 /// </summary>
00062         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00063         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00064
00065 /// <summary>
00066 /// Presentation attributes for the slices.  An element from this collection is used for each slice in
00067 /// the pie/doughnut; if there are more slices than elements in this collection, the presentation
      attributes
00068 /// are wrapped.
00069 /// </summary>
00070         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00071
00072 /// <summary>
00073 /// A tag to identify the pie/doughnut in the plot.
00074 /// </summary>
00075         public string Tag { get; set; }
00076
00077 /// <summary>
00078 /// Create a new <see cref="Pie"/> instance drawing a pie chart.
00079 /// </summary>
00080 /// <param name="data">The data in the pie chart.  The values do not need to be normalised.</param>
00081 /// <param name="centre">The centre of the pie, in data coordinates.</param>
00082 /// <param name="radius">The radius of the pie, in data coordinates.</param>
00083 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00084         public Pie(IReadOnlyList<double> data, IReadOnlyList<double> centre, IReadOnlyList<double>
      radius, ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00085         {
00086             Data = data;
00087             Centre = centre;
00088             OuterRadius = radius;
00089             InnerRadius = new double[] { 0, 0 };
00090             CoordinateSystem = coordinateSystem;
00091         }
00092
00093 /// <summary>
00094 /// Create a new <see cref="Pie"/> instance drawing a doughnut chart.
00095 /// </summary>
00096 /// <param name="data">The data in the doughnut chart.  The values do not need to be
      normalised.</param>
00097 /// <param name="centre">The centre of the doughnut, in data coordinates.</param>
00098 /// <param name="innerRadius">The inner radius of the doughnut, in data coordinates.</param>
00099 /// <param name="outerRadius">The outer radius of the doughnut, in data coordinates.</param>
00100 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00101         public Pie(IReadOnlyList<double> data, IReadOnlyList<double> centre, IReadOnlyList<double>
      innerRadius, IReadOnlyList<double> outerRadius, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem)
00102         {
00103             Data = data;
00104             Centre = centre;
00105             OuterRadius = outerRadius;
00106             InnerRadius = innerRadius;
00107             CoordinateSystem = coordinateSystem;
00108         }
00109
00110 /// <inheritdoc/>
00111         public void Plot(Graphics target)
00112         {
00113             double total = Data.Sum();
00114
00115             double currAngle = StartAngle;
00116
00117             double minAngle = Data.Where(x => x > 0).Min() / total * 2 * Math.PI;
00118
00119             minAngle = Math.Min(minAngle, 1);
00120             minAngle = Math.Max(minAngle, 0.1);
00121
00122             for (int i = 0; i < Data.Count; i++)
00123             {
00124                 double endAngle = currAngle + Data[i] / total * 2 * Math.PI;
00125
00126                 GraphicsPath pth = new GraphicsPath();
00127
00128                 if (InnerRadius[0] != 0 || InnerRadius[1] != 0)
00129                 {
00130                     Point[] innerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00131
00132                     Point[] outerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00133
```

```
00134                     double step = (endAngle - currAngle) / (innerCircle.Length - 1);
00135
00136                     for (int j = 0; j < innerCircle.Length; j++)
00137                     {
00138                         double currAng = currAngle + step * j;
00139
00140                         if (Clockwise)
00141                         {
00142                             currAng *= -1;
00143                         }
00144
00145                         innerCircle[j] = CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] +
      InnerRadius[0] * Math.Cos(currAng), Centre[1] + InnerRadius[1] * Math.Sin(currAng) });
00146                         outerCircle[outerCircle.Length - 1 - j] =
      CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] + OuterRadius[0] * Math.Cos(currAng),
      Centre[1] + OuterRadius[1] * Math.Sin(currAng) });
00147                     }
00148
00149                     currAngle = endAngle;
00150
00151                     pth.AddSmoothSpline(innerCircle);
00152
00153                     pth.LineTo(outerCircle[0]);
00154                     pth.AddSmoothSpline(outerCircle);
00155
00156                     pth.Close();
00157                 }
00158                 else
00159                 {
00160                     Point[] outerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00161
00162                     double step = (endAngle - currAngle) / (outerCircle.Length - 1);
00163
00164                     for (int j = 0; j < outerCircle.Length; j++)
00165                     {
00166                         double currAng = currAngle + step * j;
00167
00168                         if (Clockwise)
00169                         {
00170                             currAng *= -1;
00171                         }
00172
00173                         outerCircle[outerCircle.Length - 1 - j] =
      CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] + OuterRadius[0] * Math.Cos(currAng),
      Centre[1] + OuterRadius[1] * Math.Sin(currAng) });
00174                     }
00175
00176                     currAngle = endAngle;
00177
00178                     pth.MoveTo(CoordinateSystem.ToPlotCoordinates(Centre));
00179
00180                     pth.LineTo(outerCircle[0]);
00181                     pth.AddSmoothSpline(outerCircle);
00182
00183                     pth.Close();
00184                 }
00185
00186                 PlotElementPresentationAttributes attributes = PresentationAttributes[i %
      PresentationAttributes.Count];
00187
00188                 string tag = Tag;
00189                 string strokeTag = tag;
00190
00191                 if (target.UseUniqueTags && !string.IsNullOrEmpty(tag))
00192                 {
00193                     tag += "@" + i.ToString();
00194                     strokeTag += "@" + i.ToString() + "_stroke";
00195                 }
00196
00197                 if (attributes.Fill != null)
00198                 {
00199                     target.FillPath(pth, attributes.Fill, tag);
00200                 }
00201
00202                 if (attributes.Stroke != null)
00203                 {
00204                     target.StrokePath(pth, attributes.Stroke, attributes.LineWidth,
      attributes.LineCap, attributes.LineJoin, attributes.LineDash, strokeTag);
00205                 }
00206             }
00207         }
00208     }
00209 }
```

8.28 Plot.Area.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new area chart.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="vertical">If this is <see langword="true"/> (the default), the highlighted area goes
      from the X axis up to the sampled values.  If this is <see langword="false"/>, the highlighted area
      goes from the Y axis to the sampled values.</param>
00033 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00045 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the area chart.</returns>
00048             public static Plot AreaChart(IReadOnlyList<IReadOnlyList<double>> data, bool vertical =
      true, bool smooth = false, double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061                 if (axisPresentationAttributes == null)
00062                 {
00063                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00064                 }
00065
00066                 if (gridPresentationAttributes == null)
00067                 {
00068                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00069                 }
00070
00071                 if (axisLabelPresentationAttributes == null)
00072                 {
00073                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00074                 }
00075
00076                 if (axisTitlePresentationAttributes == null)
00077                 {
```

```
00078                   axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00079               }
00080
00081               if (dataPresentationAttributes == null)
00082               {
00083                   dataPresentationAttributes = new PlotElementPresentationAttributes()
00084                   {
00085                       Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
00086                       Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178))
00087                   };
00088               }
00089
00090               if (titlePresentationAttributes == null)
00091               {
00092                   titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00093               }
00094
00095               double baselineValue = 0;
00096
00097               if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00098               {
00099                   baselineValue = 1;
00100               }
00101
00102               Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline;
00103
00104               if (vertical)
00105               {
00106                   getBaseline = x => new double[] { x[0], baselineValue };
00107               }
00108               else
00109               {
00110                   getBaseline = x => new double[] { baselineValue, x[1] };
00111               }
00112
00113               IReadOnlyList<double>[] allData = new IReadOnlyList<double>[data.Count * 2];
00114
00115               for (int i = 0; i < data.Count; i++)
00116               {
00117                   allData[i] = data[i];
00118                   allData[i + data.Count] = getBaseline(data[i]);
00119               }
00120
00121               (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00122
00123               if (coordinateSystem == null)
00124               {
00125                   coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00126               }
00127
00128               Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00129               Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00130               Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00131               Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00132
00133               double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00134               double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00135               double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00136               double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00137
00138               double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00139               double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00140               double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00141               double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
```

```
00142
00143
00144                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00145                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00146                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00147                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00148
00149                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00150                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00151
00152                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00153                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00154
00155                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00156                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00157
00158                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00159                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00160
00161                 Graphics xLabelsSize = new Graphics();
00162                 xLabels.Plot(xLabelsSize);
00163                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00164
00165                 Graphics yLabelsSize = new Graphics();
00166                 yLabels.Plot(yLabelsSize);
00167                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00168
00169                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00170                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00171
00172                 Area<IReadOnlyList<double>> area = new Area<IReadOnlyList<double>>(data, getBaseline,
      coordinateSystem) { PresentationAttributes = dataPresentationAttributes, Smooth = smooth };
00173
00174                 TextLabel<IReadOnlyList<double>> titleLabel = new
      TextLabel<IReadOnlyList<double>>(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00175
00176                 Plot tbr = new Plot();
00177                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle, area, titleLabel);
00178
00179                 return tbr;
00180             }
00181
00182
00183 /// <summary>
00184 /// Create a new stacked area chart.
00185 /// </summary>
00186 /// <param name="data">The data to plot.</param>
00187 /// <param name="vertical">If this is <see langword="true"/> (the default), the highlighted area goes
      from the X axis up to the sampled values.  If this is <see langword="false"/>, the highlighted area
      goes from the Y axis to the sampled values.</param>
00188 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00189 /// <param name="width">The width of the plot.</param>
00190 /// <param name="height">The height of the plot.</param>
00191 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00192 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00193 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00194 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00195 /// <param name="xAxisTitle">Title for the X axis.</param>
00196 /// <param name="yAxisTitle">Title for the Y axis.</param>
00197 /// <param name="title">Title for the plot.</param>
```

```
00198 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00199 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00200 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00201 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00202 /// <returns>A <see cref="Plot"/> containing the stacked area chart.</returns>
00203             public static Plot StackedAreaChart(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, bool smooth = false, double width = 350, double height = 250,
00204                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00205                 double axisArrowSize = 3,
00206                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00207                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00208                 string xAxisTitle = null,
00209                 string yAxisTitle = null,
00210                 string title = null,
00211                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00212                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00213                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00214                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00215             {
00216                 if (axisPresentationAttributes == null)
00217                 {
00218                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00219                 }
00220 
00221                 if (gridPresentationAttributes == null)
00222                 {
00223                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00224                 }
00225 
00226                 if (axisLabelPresentationAttributes == null)
00227                 {
00228                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00229                 }
00230 
00231                 if (axisTitlePresentationAttributes == null)
00232                 {
00233                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00234                 }
00235 
00236                 if (dataPresentationAttributes == null)
00237                 {
00238                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00239                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)),
      Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00240                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
      Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00241                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
      Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00242                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00243                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00244                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00245                 }
00246 
00247                 if (titlePresentationAttributes == null)
00248                 {
00249                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00250                 }
00251 
00252                 double baselineValue = 0;
00253 
00254                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00255                 {
00256                     baselineValue = 1;
00257                 }
00258 
00259                 Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline;
00260 
```

```
00261                 if (vertical)
00262                 {
00263                     getBaseline = x => new double[] { x[0], baselineValue };
00264                 }
00265                 else
00266                 {
00267                     getBaseline = x => new double[] { baselineValue, x[1] };
00268                 }
00269 
00270                 List<IReadOnlyList<double» allData = new List<IReadOnlyList<double»(data.Count *
      data[0].Count);
00271 
00272                 allData.AddRange(data);
00273 
00274                 List<List<IReadOnlyList<double»> groupedData = new
      List<List<IReadOnlyList<double»>(data[0].Count - 1);
00275 
00276                 List<Dictionary<double, double» baselines = new List<Dictionary<double,
      double»(data[0].Count - 2);
00277 
00278                 for (int i = 0; i < data.Count; i++)
00279                 {
00280                     allData.Add(getBaseline(data[i]));
00281                 }
00282 
00283                 for (int i = 0; i < data[0].Count - 1; i++)
00284                 {
00285                     if (vertical)
00286                     {
00287                         groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[0], el[i + 1] }));
00288 
00289                         if (i > 0)
00290                         {
00291                             Dictionary<double, double> baseline = new Dictionary<double,
      double>(data.Count);
00292 
00293                             for (int j = 0; j < data.Count; j++)
00294                             {
00295                                 baseline[data[j][0]] = data[j][i];
00296                             }
00297 
00298                             baselines.Add(baseline);
00299                         }
00300                     }
00301                     else
00302                     {
00303                         if (i == 0)
00304                         {
00305                             groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[0], el[1] }));
00306                         }
00307                         else
00308                         {
00309                             groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[i + 1], el[1] }));
00310 
00311                             Dictionary<double, double> baseline = new Dictionary<double,
      double>(data.Count);
00312 
00313                             if (i == 1)
00314                             {
00315                                 for (int j = 0; j < data.Count; j++)
00316                                 {
00317                                     baseline[data[j][1]] = data[j][0];
00318                                 }
00319                             }
00320                             else
00321                             {
00322                                 for (int j = 0; j < data.Count; j++)
00323                                 {
00324                                     baseline[data[j][1]] = data[j][i];
00325                                 }
00326                             }
00327 
00328                             baselines.Add(baseline);
00329                         }
00330                     }
00331 
00332                     allData.AddRange(groupedData[groupedData.Count - 1]);
00333                 }
00334 
00335                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00336 
00337                 if (coordinateSystem == null)
00338                 {
```

```
00339                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00340                 }
00341
00342                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00343                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00344                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00345                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00346
00347                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00348                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00349                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00350                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00351
00352                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00353                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00354                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00355                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00356
00357
00358                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00359                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00360                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00361                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00362
00363                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00364                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00365
00366                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00367                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00368
00369                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00370                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00371
00372                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00373                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00374
00375                 Graphics xLabelsSize = new Graphics();
00376                 xLabels.Plot(xLabelsSize);
00377                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00378
00379                 Graphics yLabelsSize = new Graphics();
00380                 yLabels.Plot(yLabelsSize);
00381                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00382
00383                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00384                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00385
00386
00387                 List<IPlotElement> areas = new List<IPlotElement>(groupedData.Count);
```

```
00388                 List<IPlotElement> lines = new List<IPlotElement>(groupedData.Count);
00389
00390                 for (int i = 0; i < groupedData.Count; i++)
00391                 {
00392                     if (i == 0)
00393                     {
00394                         Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], getBaseline, coordinateSystem) { PresentationAttributes =
      new PlotElementPresentationAttributes(dataPresentationAttributes[i %
      dataPresentationAttributes.Count]) { Stroke = null }, Smooth = smooth };
00395                         DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00396                         areas.Add(area);
00397                         lines.Add(line);
00398                     }
00399                     else
00400                     {
00401                         int index = i;
00402
00403                         if (vertical)
00404                         {
00405                             Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], x => new double[] { x[0], baselines[index - 1][x[0]] },
      coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Stroke = null }, Smooth = smooth };
00406                             DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00407                             areas.Add(area);
00408                             lines.Add(line);
00409                         }
00410                         else
00411                         {
00412                             Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], x => new double[] { baselines[index - 1][x[1]], x[1] },
      coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Stroke = null }, Smooth = smooth };
00413                             DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00414                             areas.Add(area);
00415                             lines.Add(line);
00416                         }
00417                     }
00418
00419                 }
00420
00421                 TextLabel<IReadOnlyList<double>> titleLabel = new
      TextLabel<IReadOnlyList<double>>(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00422
00423                 Plot tbr = new Plot();
00424                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00425                 tbr.AddPlotElements(areas);
00426                 tbr.AddPlotElements(lines);
00427                 tbr.AddPlotElement(titleLabel);
00428
00429                 return tbr;
00430             }
00431         }
00432     }
00433 }
```

8.29 Plot.BarChart.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new bar chart.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00033 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00045 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00048             public static Plot BarChart(IReadOnlyList<double> data, bool vertical = true, double
      margin = 0.25, double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061                 double[][] changedData = new double[data.Count][];
00062
00063                 if (vertical)
00064                 {
00065                     for (int i = 0; i < data.Count; i++)
00066                     {
00067                         changedData[i] = new double[] { i, data[i] };
00068                     }
00069                 }
00070                 else
00071                 {
00072                     for (int i = 0; i < data.Count; i++)
00073                     {
00074                         changedData[i] = new double[] { data[i], i };
00075                     }
00076                 }
00077
00078                 Plot barChart = Histogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00079
00080                 if (vertical)
00081                 {
00082                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00083                 }
00084                 else
00085                 {
00086                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00087                 }
00088
00089                 Plot tbr = new Plot();
```

```
00090
00091                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00092                 {
00093                     if ((vertical && i != 5 && i != 3 && i != 1) || (!vertical && i != 6 && i != 4 &&
      i != 2))
00094                     {
00095                         tbr.AddPlotElement(barChart.PlotElements[i]);
00096                     }
00097                 }
00098
00099                 return tbr;
00100             }
00101
00102 /// <summary>
00103 /// Create a new bar chart.
00104 /// </summary>
00105 /// <param name="data">The data to plot.</param>
00106 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00107 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00108 /// <param name="width">The width of the plot.</param>
00109 /// <param name="height">The height of the plot.</param>
00110 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00111 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00112 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00113 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00114 /// <param name="xAxisTitle">Title for the X axis.</param>
00115 /// <param name="yAxisTitle">Title for the Y axis.</param>
00116 /// <param name="title">Title for the plot.</param>
00117 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00118 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00119 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00120 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00121 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00122             public static Plot BarChart<T>(IReadOnlyList<(T, double)> data, bool vertical = true,
      double margin = 0.25, double width = 350, double height = 250,
00123                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00124                 double axisArrowSize = 3,
00125                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00126                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00127                 string xAxisTitle = null,
00128                 string yAxisTitle = null,
00129                 string title = null,
00130                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00131                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00132                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00133                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00134             {
00135                 return BarChart(data, double.NaN, double.NaN, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00136             }
00137
00138
00139             private static Plot BarChart<T>(IReadOnlyList<(T, double)> data, double overrideMax,
      double overrideMin, bool vertical = true, double margin = 0.25, double width = 350, double height =
      250,
00140                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00141                 double axisArrowSize = 3,
00142                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00143                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00144                 string xAxisTitle = null,
00145                 string yAxisTitle = null,
00146                 string title = null,
00147                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00148                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00149                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00150                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00151             {
00152                 double[][] changedData = new double[data.Count][];
00153
00154                 if (vertical)
00155                 {
00156                     for (int i = 0; i < data.Count; i++)
00157                     {
00158                         changedData[i] = new double[] { i, data[i].Item2 };
00159                     }
00160                 }
00161                 else
00162                 {
00163                     for (int i = 0; i < data.Count; i++)
00164                     {
00165                         changedData[i] = new double[] { data[i].Item2, i };
```

```
00166                     }
00167                 }
00168 
00169                 Plot barChart = Histogram(changedData, overrideMax, overrideMin, vertical, margin,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00170 
00171                 if (vertical)
00172                 {
00173                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00174                 }
00175                 else
00176                 {
00177                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00178                 }
00179 
00180                 Plot tbr = new Plot();
00181 
00182                 double yLabelsWidth = 0;
00183                 double xLabelsHeight = 0;
00184 
00185                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00186                 {
00187                     if ((vertical && i != 5) || (!vertical && i != 6))
00188                     {
00189                         tbr.AddPlotElement(barChart.PlotElements[i]);
00190                     }
00191                     else
00192                     {
00193                         if (vertical)
00194                         {
00195                             ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00196 
00197                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00198 
00199                             for (int j = 0; j < data.Count; j++)
00200                             {
00201                                 labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 - (double)j
      / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
00202                             }
00203 
00204                             DataLabels<IReadOnlyList<double» xLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
00205 
00206                             Graphics xLabelsSize = new Graphics();
00207                             xLabels.Plot(xLabelsSize);
00208                             xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00209 
00210                             tbr.AddPlotElement(xLabels);
00211                         }
00212                         else
00213                         {
00214                             ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
00215 
00216                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00217 
00218                             for (int j = 0; j < data.Count; j++)
00219                             {
00220                                 labelData[j] = new double[] { yAxis.StartPoint[0] * (1 - (double)j /
      (data.Count - 1)) + yAxis.EndPoint[0] * j / (data.Count - 1), j };
00221                             }
00222 
00223                             DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
00224 
00225                             Graphics yLabelsSize = new Graphics();
00226                             yLabels.Plot(yLabelsSize);
00227                             yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00228 
00229                             tbr.AddPlotElement(yLabels);
00230                         }
00231                     }
00232                 }
00233 
00234                 if (vertical)
00235                 {
00236                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
00237                 }
00238                 else
00239                 {
00240                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
```

```
00241                 }
00242
00243                 return tbr;
00244             }
00245
00246 /// <summary>
00247 /// Create a new stacked bar chart.
00248 /// </summary>
00249 /// <param name="data">The data to plot.</param>
00250 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00251 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00252 /// <param name="width">The width of the plot.</param>
00253 /// <param name="height">The height of the plot.</param>
00254 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00255 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00256 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00257 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00258 /// <param name="xAxisTitle">Title for the X axis.</param>
00259 /// <param name="yAxisTitle">Title for the Y axis.</param>
00260 /// <param name="title">Title for the plot.</param>
00261 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00262 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00263 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00264 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00265 /// <returns>A <see cref="Plot"/> containing the stacked bar chart.</returns>
00266             public static Plot StackedBarChart(IReadOnlyList<IReadOnlyList<double» data, bool vertical
      = true, double margin = 0.25, double width = 350, double height = 250,
00267                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00268                 double axisArrowSize = 3,
00269                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00270                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00271                 string xAxisTitle = null,
00272                 string yAxisTitle = null,
00273                 string title = null,
00274                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00275                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00276                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00277                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00278             {
00279                 double[][] changedData = new double[data.Count][];
00280
00281                 if (vertical)
00282                 {
00283                     for (int i = 0; i < data.Count; i++)
00284                     {
00285                         changedData[i] = new double[data[i].Count + 1];
00286
00287                         changedData[i][0] = i;
00288
00289                         for (int j = 0; j < data[i].Count; j++)
00290                         {
00291                             changedData[i][j + 1] = data[i][j];
00292                         }
00293                     }
00294                 }
00295                 else
00296                 {
00297                     for (int i = 0; i < data.Count; i++)
00298                     {
00299                         changedData[i] = new double[data[i].Count + 1];
00300
00301                         changedData[i][1] = i;
00302
00303                         changedData[i][0] = data[i][0];
00304
00305                         for (int j = 1; j < data[i].Count; j++)
00306                         {
00307                             changedData[i][j + 1] = data[i][j];
00308                         }
00309                     }
00310                 }
00311
00312                 Plot barChart = StackedHistogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00313
00314                 if (vertical)
00315                 {
00316                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00317                 }
00318                 else
00319                 {
```

```
00320                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00321                 }
00322 
00323                 Plot tbr = new Plot();
00324 
00325                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00326                 {
00327                     if ((vertical && i != 5 && i != 3 && i != 1) || (!vertical && i != 6 && i != 4 &&
      i != 2))
00328                     {
00329                         tbr.AddPlotElement(barChart.PlotElements[i]);
00330                     }
00331 
00332                 }
00333 
00334                 return tbr;
00335             }
00336 
00337 /// <summary>
00338 /// Create a new stacked bar chart.
00339 /// </summary>
00340 /// <param name="data">The data to plot.</param>
00341 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00342 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00343 /// <param name="width">The width of the plot.</param>
00344 /// <param name="height">The height of the plot.</param>
00345 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00346 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00347 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00348 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00349 /// <param name="xAxisTitle">Title for the X axis.</param>
00350 /// <param name="yAxisTitle">Title for the Y axis.</param>
00351 /// <param name="title">Title for the plot.</param>
00352 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00353 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00354 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00355 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00356 /// <returns>A <see cref="Plot"/> containing the stacked bar chart.</returns>
00357             public static Plot StackedBarChart<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, bool
      vertical = true, double margin = 0.25, double width = 350, double height = 250,
00358                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00359                 double axisArrowSize = 3,
00360                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00361                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00362                 string xAxisTitle = null,
00363                 string yAxisTitle = null,
00364                 string title = null,
00365                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00366                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00367                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00368                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00369             {
00370                 double[][] changedData = new double[data.Count][];
00371 
00372                 if (vertical)
00373                 {
00374                     for (int i = 0; i < data.Count; i++)
00375                     {
00376                         changedData[i] = new double[data[i].Item2.Count + 1];
00377 
00378                         changedData[i][0] = i;
00379 
00380                         for (int j = 0; j < data[i].Item2.Count; j++)
00381                         {
00382                             changedData[i][j + 1] = data[i].Item2[j];
00383                         }
00384                     }
00385                 }
00386                 else
00387                 {
00388                     for (int i = 0; i < data.Count; i++)
00389                     {
00390                         changedData[i] = new double[data[i].Item2.Count + 1];
00391 
00392                         changedData[i][1] = i;
00393 
00394                         changedData[i][0] = data[i].Item2[0];
00395 
00396                         for (int j = 1; j < data[i].Item2.Count; j++)
00397                         {
00398                             changedData[i][j + 1] = data[i].Item2[j];
00399                         }
00400                     }
```

```
00401                 }
00402
00403                 Plot barChart = StackedHistogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00404
00405                 if (vertical)
00406                 {
00407                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00408                 }
00409                 else
00410                 {
00411                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00412                 }
00413
00414                 Plot tbr = new Plot();
00415
00416                 double xLabelsHeight = 0;
00417                 double yLabelsWidth = 0;
00418
00419                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00420                 {
00421                     if ((vertical && i != 5) || (!vertical && i != 6))
00422                     {
00423                         tbr.AddPlotElement(barChart.PlotElements[i]);
00424                     }
00425                     else
00426                     {
00427                         if (vertical)
00428                         {
00429                             ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00430
00431                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00432
00433                             for (int j = 0; j < data.Count; j++)
00434                             {
00435                                 labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 - (double)j
      / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
00436                             }
00437
00438                             DataLabels<IReadOnlyList<double» xLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
00439
00440                             Graphics xLabelsSize = new Graphics();
00441                             xLabels.Plot(xLabelsSize);
00442                             xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00443
00444
00445                             tbr.AddPlotElement(xLabels);
00446                         }
00447                         else
00448                         {
00449                             ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
00450
00451                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00452
00453                             for (int j = 0; j < data.Count; j++)
00454                             {
00455                                 labelData[j] = new double[] { yAxis.StartPoint[0] * (1 - (double)j /
      (data.Count - 1)) + yAxis.EndPoint[0] * j / (data.Count - 1), j };
00456                             }
00457
00458                             DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
00459
00460                             Graphics yLabelsSize = new Graphics();
00461                             yLabels.Plot(yLabelsSize);
00462                             yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00463
00464                             tbr.AddPlotElement(yLabels);
00465                         }
00466                     }
00467                 }
00468
00469                 if (vertical)
00470                 {
00471                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
00472                 }
00473                 else
00474                 {
00475                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
```

```
00476                 }
00477
00478
00479                 return tbr;
00480             }
00481
00482 /// <summary>
00483 /// Create a new clustered bar chart.
00484 /// </summary>
00485 /// <param name="data">The data to plot.</param>
00486 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00487 /// <param name="interClusterMargin">Spacing between consecutive bar clusters.  This should be a value
      between 0 and 1.</param>
00488 /// <param name="intraClusterMargin">Spacing between consecutive bars within the same clusters.  This
      should be a value between 0 and 1.</param>
00489 /// <param name="width">The width of the plot.</param>
00490 /// <param name="height">The height of the plot.</param>
00491 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00492 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00493 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00494 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00495 /// <param name="xAxisTitle">Title for the X axis.</param>
00496 /// <param name="yAxisTitle">Title for the Y axis.</param>
00497 /// <param name="title">Title for the plot.</param>
00498 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00499 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00500 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00501 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00502 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00503             public static Plot ClusteredBarChart(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, double interClusterMargin = 0.25, double intraClusterMargin = 0, double width = 350,
      double height = 250,
00504                     PlotElementPresentationAttributes axisPresentationAttributes = null,
00505                     double axisArrowSize = 3,
00506                     PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00507                     PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00508                     string xAxisTitle = null,
00509                     string yAxisTitle = null,
00510                     string title = null,
00511                     PlotElementPresentationAttributes titlePresentationAttributes = null,
00512                     PlotElementPresentationAttributes gridPresentationAttributes = null,
00513                     IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes =
      null,
00514                     IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00515             {
00516                 if (axisPresentationAttributes == null)
00517                 {
00518                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00519                 }
00520
00521                 if (gridPresentationAttributes == null)
00522                 {
00523                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00524                 }
00525
00526                 if (axisLabelPresentationAttributes == null)
00527                 {
00528                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00529                 }
00530
00531                 if (axisTitlePresentationAttributes == null)
00532                 {
00533                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00534                 }
00535
00536                 if (dataPresentationAttributes == null)
00537                 {
00538                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00539                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00540                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00541                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00542                                                                                            new
```

```
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
      180, 233)) },
00543                                                                                 new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00544                                                                                 new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00545                 }
00546
00547                 double baselineValue = 0;
00548
00549                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00550                 {
00551                     baselineValue = 1;
00552                 }
00553
00554                 ClusteredBars bars;
00555
00556                 if (baselineValue == 0)
00557                 {
00558                     bars = new ClusteredBars(data, coordinateSystem, vertical) {
      PresentationAttributes = dataPresentationAttributes, InterClusterMargin = interClusterMargin,
      IntraClusterMargin = intraClusterMargin };
00559                 }
00560                 else
00561                 {
00562                     bars = new ClusteredBars(data, new Comparison<IReadOnlyList<double»(vertical ?
      new Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x,
      y) => Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double»(pt =>
      new double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] },
      coordinateSystem) { PresentationAttributes = dataPresentationAttributes, InterClusterMargin =
      interClusterMargin, IntraClusterMargin = intraClusterMargin };
00563                 }
00564
00565                 double minX, minY, maxX, maxY, rangeX, rangeY;
00566
00567                 {
00568                     minX = double.MaxValue;
00569                     minY = double.MaxValue;
00570
00571                     maxX = double.MinValue;
00572                     maxY = double.MinValue;
00573
00574                     for (int i = 0; i < data.Count; i++)
00575                     {
00576                         minX = Math.Min(minX, data[i][0]);
00577                         minY = Math.Min(minY, data[i][1]);
00578
00579                         maxX = Math.Max(maxX, data[i][0]);
00580                         maxY = Math.Max(maxY, data[i][1]);
00581
00582                         if (vertical)
00583                         {
00584                             for (int j = 2; j < data[i].Count; j++)
00585                             {
00586                                 minY = Math.Min(minY, data[i][j]);
00587                                 maxY = Math.Max(maxY, data[i][j]);
00588                             }
00589                         }
00590                         else
00591                         {
00592                             for (int j = 2; j < data[i].Count; j++)
00593                             {
00594                                 minX = Math.Min(minX, data[i][j]);
00595                                 maxX = Math.Max(maxX, data[i][j]);
00596                             }
00597                         }
00598                     }
00599
00600                     rangeX = maxX - minX;
00601                     rangeY = maxY - minY;
00602                 }
00603
00604                 double dataMinX = minX;
00605                 double dataMinY = minY;
00606                 double dataMaxX = maxX;
00607                 double dataMaxY = maxY;
00608
00609                 if (coordinateSystem == null)
00610                 {
00611                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
00612
00613                     if (vertical)
```

```
00614                     {
00615                         minY = Math.Min(minY, 0);
00616                         maxY = Math.Max(maxY, 0);
00617                         rangeY = maxY - minY;
00618 
00619                         linCoords.MinY = minY - rangeY * 0.1;
00620                         linCoords.MaxY = maxY + rangeY * 0.1;
00621 
00622 
00623                         if (bars.Data.Count >= 2)
00624                         {
00625                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00626                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00627 
00628                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00629                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00630 
00631                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00632                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00633 
00634                             rangeX = maxX - minX;
00635                             linCoords.MinX = minX;
00636                             linCoords.MaxX = maxX;
00637                         }
00638                     }
00639                     else
00640                     {
00641                         minX = Math.Min(minX, 0);
00642                         maxX = Math.Max(maxX, 0);
00643                         rangeX = maxX - minX;
00644 
00645                         linCoords.MinX = minX - rangeX * 0.1;
00646                         linCoords.MaxX = maxX + rangeX * 0.1;
00647 
00648 
00649                         if (bars.Data.Count >= 2)
00650                         {
00651                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00652                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00653 
00654                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00655                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00656 
00657                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00658                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00659 
00660                             rangeY = maxY - minY;
00661                             linCoords.MinY = minY;
00662                             linCoords.MaxY = maxY;
00663                         }
00664                     }
00665 
00666                     coordinateSystem = linCoords;
00667                     bars.CoordinateSystem = coordinateSystem;
00668                 }
00669                 else
00670                 {
00671                     if (vertical)
00672                     {
00673                         minY = Math.Min(minY, baselineValue);
00674                         rangeY = maxY - minY;
00675 
00676                         if (bars.Data.Count >= 2)
00677                         {
00678                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00679                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00680 
00681                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00682                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00683 
00684                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00685                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00686 
00687                             rangeX = maxX - minX;
00688                         }
00689                     }
00690                     else
00691                     {
00692                         minX = Math.Min(minX, baselineValue);
00693                         rangeX = maxX - minX;
00694 
00695                         if (bars.Data.Count >= 2)
00696                         {
00697                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00698                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00699 
00700                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
```

```
00701                               IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00702
00703                               minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00704                               maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00705
00706                               rangeY = maxY - minY;
00707                           }
00708                       }
00709                   }
00710
00711
00712
00713                   if (titlePresentationAttributes == null)
00714                   {
00715                       titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00716                   }
00717
00718                   Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00719                   Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00720                   Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00721                   Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00722
00723                   double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00724                   double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00725                   double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00726                   double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00727
00728                   double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00729                   double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00730                   double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00731                   double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00732
00733
00734                   double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00735                   double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00736                   double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00737                   double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00738
00739                   if (vertical)
00740                   {
00741                       Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00742                       Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
00743
00744                       Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00745                       Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00746
00747                       Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00748                       Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
00749
00750                       Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00751                       Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
```

```
00752
00753                   p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
00754                   p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
00755                   p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
00756                   p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
00757               }
00758               else
00759               {
00760                   Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00761                   Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
00762
00763                   Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00764                   Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
00765
00766                   Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00767                   Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00768
00769                   Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00770                   Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
      });
00771
00772                   p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
00773                   p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
00774                   p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
00775                   p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
00776               }
00777
00778               Grid grid;
00779
00780               if (vertical)
00781               {
00782                   grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00783               }
00784               else
00785               {
00786                   grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00787               }
00788
00789               ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00790               ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00791
00792               int every = (int)Math.Ceiling((double)data.Count / 5);
00793               int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
00794
00795               ContinuousAxisTicks xTicks;
00796               ContinuousAxisTicks yTicks;
00797
00798               if (vertical)
00799               {
00800                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00801                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00802               }
00803               else
00804               {
00805                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00806                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00807               }
00808
00809               ContinuousAxisLabels xLabels;
00810               ContinuousAxisLabels yLabels;
00811
00812               if (vertical)
00813               {
00814                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
00815                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
```

```
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00816
00817                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      xLabels.TextFormat;
00818                     xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00819                 }
00820                 else
00821                 {
00822                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00823                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
00824
00825                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      yLabels.TextFormat;
00826                     yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00827                 }
00828
00829                 Graphics xLabelsSize = new Graphics();
00830                 xLabels.Plot(xLabelsSize);
00831                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00832
00833                 Graphics yLabelsSize = new Graphics();
00834                 yLabels.Plot(yLabelsSize);
00835                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00836
00837                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00838                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00839
00840                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00841
00842                 Plot tbr = new Plot();
00843                 tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
      yTitle, bars, titleLabel);
00844
00845                 return tbr;
00846             }
00847
00848 /// <summary>
00849 /// Create a new clustered bar chart.
00850 /// </summary>
00851 /// <param name="data">The data to plot.</param>
00852 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00853 /// <param name="interClusterMargin">Spacing between consecutive bar clusters.  This should be a value
      between 0 and 1.</param>
00854 /// <param name="intraClusterMargin">Spacing between consecutive bars within the same clusters.  This
      should be a value between 0 and 1.</param>
00855 /// <param name="width">The width of the plot.</param>
00856 /// <param name="height">The height of the plot.</param>
00857 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00858 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00859 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00860 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00861 /// <param name="xAxisTitle">Title for the X axis.</param>
00862 /// <param name="yAxisTitle">Title for the Y axis.</param>
00863 /// <param name="title">Title for the plot.</param>
00864 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00865 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00866 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00867 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00868 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00869             public static Plot ClusteredBarChart<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data,
      bool vertical = true, double interClusterMargin = 0.25, double intraClusterMargin = 0, double width =
      350, double height = 250,
00870                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00871                 double axisArrowSize = 3,
00872                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00873                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00874                 string xAxisTitle = null,
00875                 string yAxisTitle = null,
00876                 string title = null,
00877                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00878                 PlotElementPresentationAttributes gridPresentationAttributes = null,
```

```
00879               IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00880               IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00881           {
00882               double[][] changedData = new double[data.Count][];
00883
00884               if (vertical)
00885               {
00886                   for (int i = 0; i < data.Count; i++)
00887                   {
00888                       changedData[i] = new double[data[i].Item2.Count + 1];
00889
00890                       changedData[i][0] = i;
00891
00892                       for (int j = 0; j < data[i].Item2.Count; j++)
00893                       {
00894                           changedData[i][j + 1] = data[i].Item2[j];
00895                       }
00896                   }
00897               }
00898               else
00899               {
00900                   for (int i = 0; i < data.Count; i++)
00901                   {
00902                       changedData[i] = new double[data[i].Item2.Count + 1];
00903
00904                       changedData[i][1] = i;
00905
00906                       changedData[i][0] = data[i].Item2[0];
00907
00908                       for (int j = 1; j < data[i].Item2.Count; j++)
00909                       {
00910                           changedData[i][j + 1] = data[i].Item2[j];
00911                       }
00912                   }
00913               }
00914
00915               Plot barChart = ClusteredBarChart(changedData, vertical, interClusterMargin,
      intraClusterMargin, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes,
      coordinateSystem);
00916
00917               if (vertical)
00918               {
00919                   ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00920               }
00921               else
00922               {
00923                   ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00924               }
00925
00926               Plot tbr = new Plot();
00927
00928               double xLabelsHeight = 0;
00929               double yLabelsWidth = 0;
00930
00931               for (int i = 0; i < barChart.PlotElements.Count; i++)
00932               {
00933                   if ((vertical && i != 5) || (!vertical && i != 6))
00934                   {
00935                       tbr.AddPlotElement(barChart.PlotElements[i]);
00936                   }
00937                   else
00938                   {
00939                       if (vertical)
00940                       {
00941                           ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00942
00943                           IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00944
00945                           for (int j = 0; j < data.Count; j++)
00946                           {
00947                               labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 - (double)j
      / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
00948                           }
00949
00950                           DataLabels<IReadOnlyList<double» xLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
00951
00952                           Graphics xLabelsSize = new Graphics();
00953                           xLabels.Plot(xLabelsSize);
00954                           xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00955
00956                           tbr.AddPlotElement(xLabels);
00957                       }
```

```
00958                         else
00959                         {
00960                             ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
00961 
00962                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00963 
00964                             for (int j = 0; j < data.Count; j++)
00965                             {
00966                                 labelData[j] = new double[] { yAxis.StartPoint[0] * (1 - (double)j /
      (data.Count - 1)) + yAxis.EndPoint[0] * j / (data.Count - 1), j };
00967                             }
00968 
00969                             DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
00970 
00971                             Graphics yLabelsSize = new Graphics();
00972                             yLabels.Plot(yLabelsSize);
00973                             yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00974 
00975                             tbr.AddPlotElement(yLabels);
00976                         }
00977                     }
00978 
00979                 }
00980 
00981                 if (vertical)
00982                 {
00983                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
00984                 }
00985                 else
00986                 {
00987                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
00988                 }
00989 
00990                 return tbr;
00991             }
00992         }
00993     }
00994 }
```

8.30 Plot.BoxPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using MathNet.Numerics.Statistics;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 
00023 namespace VectSharp.Plots
00024 {
00025     partial class Plot
00026     {
00027 /// <summary>
00028 /// Describes the types of whiskers for a box plot.
00029 /// </summary>
00030         public enum WhiskerType
00031         {
00032 /// <summary>
00033 /// The whiskers extend from the minimum sampled value to the maximum sampled value.
00034 /// </summary>
00035             FullRange,
00036 
00037 /// <summary>
00038 /// The whiskers extend from (Q1 - 1.5 * IQR) to (Q3 + 1.5 * IQR).
```

```
00039 /// </summary>
00040             IQR_1_5,
00041
00042 /// <summary>
00043 /// The whiskers extend from (mean - 2 * standard deviation) to (mean + 2 * standard deviation).
00044 /// </summary>
00045             StandardDeviation
00046         }
00047
00048         partial class Create
00049         {
00050 /// <summary>
00051 /// Create a new box plot.
00052 /// </summary>
00053 /// <param name="data">The data to plot.</param>
00054 /// <param name="vertical">If this is <see langword="true"/> (the default), the boxes are parallel to the Y axis.  If this is <see langword="false"/>, the boxes are parallel to the X axis.</param>
00055 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00056 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has notches.  Otherwise, it does not.</param>
00057 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the boxes have the same width.  Otherwise, the width of each box is proportional to the number of samples.</param>
00058 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four outlier points that fall outside of the interval between the whiskers.</param>
00059 /// <param name="boxWidth">The width of the boxes in data space coordinates.</param>
00060 /// <param name="spacing">The spacing between consecutive boxes (ranging from 0 to 1).</param>
00061 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override the default minimum value for the plot.  Useful if you wish to create multiple plots with the same scale,
00062 /// even though the sampled values have different ranges.</param>
00063 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override the default maximum value for the plot.  Useful if you wish to create multiple plots with the same scale,
00064 /// even though the sampled values have different ranges.</param>
00065 /// <param name="width">The width of the plot.</param>
00066 /// <param name="height">The height of the plot.</param>
00067 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00068 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00069 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00070 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00071 /// <param name="xAxisTitle">Title for the X axis.</param>
00072 /// <param name="yAxisTitle">Title for the Y axis.</param>
00073 /// <param name="title">Title for the plot.</param>
00074 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00075 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00076 /// <param name="boxPresentationAttributes">Presentation attributes for the boxes.</param>
00077 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00078 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier points.</param>
00079 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00080 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00081             public static Plot BoxPlot<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, WhiskerType whiskerType = WhiskerType.IQR_1_5, bool useNotches = true, bool proportionalWidth = false, bool showOutliers = true, double boxWidth = 10, double spacing = 0.1, double?  dataRangeMin = null, double? dataRangeMax = null, bool vertical = true, double width = 350, double height = 250,
00082                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00083                 double axisArrowSize = 3,
00084                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00085                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00086                 string xAxisTitle = null,
00087                 string yAxisTitle = null,
00088                 string title = null,
00089                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00090                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00091                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00092                 IDataPointElement outlierPointElement = null,
00093                 IReadOnlyList<PlotElementPresentationAttributes> outlierPresentationAttributes = null,
00094                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00095             {
00096                 if (boxPresentationAttributes == null)
00097                 {
00098                     boxPresentationAttributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00099                                                                                           new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)), Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00100                                                                                           new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)), Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00101                                                                                           new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)), Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00102                                                                                           new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
```

```
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00103                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00104                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00105                 }
00106
00107                 if (outlierPresentationAttributes == null)
00108                 {
00109                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[boxPresentationAttributes.Count];
00110
00111                     for (int i = 0; i < boxPresentationAttributes.Count; i++)
00112                     {
00113                         temp[i] = new PlotElementPresentationAttributes() { Stroke = null, Fill =
      boxPresentationAttributes[i].Stroke };
00114                     }
00115
00116                     outlierPresentationAttributes = temp;
00117                 }
00118
00119                 if (axisPresentationAttributes == null)
00120                 {
00121                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00122                 }
00123
00124                 if (gridPresentationAttributes == null)
00125                 {
00126                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00127                 }
00128
00129                 if (axisLabelPresentationAttributes == null)
00130                 {
00131                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00132                 }
00133
00134                 if (axisTitlePresentationAttributes == null)
00135                 {
00136                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00137                 }
00138
00139                 if (titlePresentationAttributes == null)
00140                 {
00141                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00142                 }
00143
00144                 if (outlierPointElement == null)
00145                 {
00146                     outlierPointElement = new PathDataPointElement();
00147                 }
00148
00149                 double[] medians = new double[data.Count];
00150                 double[] firstQuartiles = new double[data.Count];
00151                 double[] thirdQuartiles = new double[data.Count];
00152                 double[] upperWhiskers = new double[data.Count];
00153                 double[] lowerWhiskers = new double[data.Count];
00154                 double[] notches = new double[data.Count];
00155
00156                 double minData = double.MaxValue;
00157                 double maxData = double.MinValue;
00158
00159                 for (int i = 0; i < data.Count; i++)
00160                 {
00161                     medians[i] = data[i].Item2.Median();
00162                     firstQuartiles[i] = data[i].Item2.LowerQuartile();
00163                     thirdQuartiles[i] = data[i].Item2.UpperQuartile();
00164
00165                     if (useNotches)
00166                     {
00167                         notches[i] = 1.58 * (thirdQuartiles[i] - firstQuartiles[i]) /
      Math.Sqrt(data[i].Item2.Count);
00168                     }
00169                     else
00170                     {
00171                         notches[i] = 0;
00172                     }
00173
00174                     if (whiskerType == WhiskerType.FullRange)
00175                     {
```

```
00176                         double uW = double.MinValue;
00177                         double lW = double.MaxValue;
00178 
00179                         for (int j = 0; j < data[i].Item2.Count; j++)
00180                         {
00181                             uW = Math.Max(uW, data[i].Item2[j]);
00182                             lW = Math.Min(lW, data[i].Item2[j]);
00183 
00184                             minData = Math.Min(minData, data[i].Item2[j]);
00185                             maxData = Math.Max(maxData, data[i].Item2[j]);
00186                         }
00187 
00188                         upperWhiskers[i] = uW;
00189                         lowerWhiskers[i] = lW;
00190                     }
00191                     else if (whiskerType == WhiskerType.IQR_1_5)
00192                     {
00193                         upperWhiskers[i] = thirdQuartiles[i] + (thirdQuartiles[i] - firstQuartiles[i])
      * 1.5;
00194                         lowerWhiskers[i] = firstQuartiles[i] - (thirdQuartiles[i] - firstQuartiles[i])
      * 1.5;
00195 
00196                         double uW = double.MinValue;
00197                         double lW = double.MaxValue;
00198 
00199                         for (int j = 0; j < data[i].Item2.Count; j++)
00200                         {
00201                             if (data[i].Item2[j] <= upperWhiskers[i])
00202                             {
00203                                 uW = Math.Max(uW, data[i].Item2[j]);
00204                             }
00205 
00206                             if (data[i].Item2[j] >= lowerWhiskers[i])
00207                             {
00208                                 lW = Math.Min(lW, data[i].Item2[j]);
00209                             }
00210 
00211                             minData = Math.Min(minData, data[i].Item2[j]);
00212                             maxData = Math.Max(maxData, data[i].Item2[j]);
00213                         }
00214 
00215                         upperWhiskers[i] = uW;
00216                         lowerWhiskers[i] = lW;
00217                     }
00218                     else if (whiskerType == WhiskerType.StandardDeviation)
00219                     {
00220                         double stdDev = data[i].Item2.StandardDeviation();
00221                         double mean = data[i].Item2.Mean();
00222 
00223                         upperWhiskers[i] = mean + 2 * stdDev;
00224                         lowerWhiskers[i] = mean - 2 * stdDev;
00225 
00226                         double uW = double.MinValue;
00227                         double lW = double.MaxValue;
00228 
00229                         for (int j = 0; j < data[i].Item2.Count; j++)
00230                         {
00231                             if (data[i].Item2[j] <= upperWhiskers[i])
00232                             {
00233                                 uW = Math.Max(uW, data[i].Item2[j]);
00234                             }
00235 
00236                             if (data[i].Item2[j] >= lowerWhiskers[i])
00237                             {
00238                                 lW = Math.Min(lW, data[i].Item2[j]);
00239                             }
00240 
00241                             minData = Math.Min(minData, data[i].Item2[j]);
00242                             maxData = Math.Max(maxData, data[i].Item2[j]);
00243                         }
00244 
00245                         upperWhiskers[i] = uW;
00246                         lowerWhiskers[i] = lW;
00247                     }
00248                 }
00249 
00250                 if (!showOutliers)
00251                 {
00252                     minData = lowerWhiskers.Minimum();
00253                     maxData = upperWhiskers.Maximum();
00254                 }
00255 
00256                 double minX, maxX, minY, maxY;
00257 
00258                 if (vertical)
00259                 {
00260                     minX = 0;
```

```
00261                     maxX = boxWidth * (data.Count + spacing * (data.Count - 1));
00262
00263                     minY = minData;
00264                     maxY = maxData;
00265
00266                     if (dataRangeMin != null)
00267                     {
00268                         minY = dataRangeMin.Value;
00269                     }
00270
00271                     if (dataRangeMax != null)
00272                     {
00273                         maxY = dataRangeMax.Value;
00274                     }
00275                 }
00276                 else
00277                 {
00278                     minY = 0;
00279                     maxY = boxWidth * (data.Count + spacing * (data.Count - 1));
00280
00281                     minX = minData;
00282                     maxX = maxData;
00283
00284                     if (dataRangeMin != null)
00285                     {
00286                         minX = dataRangeMin.Value;
00287                     }
00288
00289                     if (dataRangeMax != null)
00290                     {
00291                         maxX = dataRangeMax.Value;
00292                     }
00293                 }
00294
00295
00296                 if (coordinateSystem == null)
00297                 {
00298                     coordinateSystem = new LinearCoordinateSystem2D(minX, maxX, minY, maxY, width,
      height);
00299                 }
00300
00301                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00302                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00303                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00304                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00305
00306                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00307                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00308                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00309                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00310
00311                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00312                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00313                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00314                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00315
00316
00317                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00318                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00319                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00320                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00321
00322                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
```

```
coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = vertical ?  0 :
axisArrowSize };
00323                   ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = vertical ?
axisArrowSize :  0 };
00324
00325                   IPlotElement xTicks;
00326                   IPlotElement yTicks;
00327
00328                   IPlotElement xLabels;
00329                   IPlotElement yLabels;
00330
00331                   if (vertical)
00332                   {
00333                       //xTicks = new ContinuousAxisTicks(new double[] { 0.5 * boxWidth, p3[1] }, new
double[] { ((data.Count - 1) * (spacing + 1) + 0.5) * boxWidth, p4[1] }, coordinateSystem) {
PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = _ =>
3, SizeBelow = _ => 3 };
00334
00335                       xTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
data.Count) select new double[] { (el * (spacing + 1) + 0.5) * boxWidth, p3[1] }), coordinateSystem) {
PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
PathDataPointElement(new GraphicsPath().MoveTo(0, -3).LineTo(0, 3)) };
00336                       yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
PresentationAttributes = axisPresentationAttributes };
00337
00338                       xLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
data.Count) select new double[] { (el * (spacing + 1) + 0.5) * boxWidth, p3[1] }), coordinateSystem) {
Alignment = TextAnchors.Center, Baseline = TextBaselines.Top, Label = (i, _) => data[i].Item1,
PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(0, 10) };
00339                       yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00340                   }
00341                   else
00342                   {
00343                       xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
PresentationAttributes = axisPresentationAttributes };
00344
00345                       yTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * boxWidth }), coordinateSystem) {
PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
PathDataPointElement(new GraphicsPath().MoveTo(-3, 0).LineTo(3, 0)) };
00346
00347                       xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00348                       yLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * boxWidth }), coordinateSystem) {
Alignment = TextAnchors.Right, Baseline = TextBaselines.Middle, Label = (i, _) => data[i].Item1,
PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(-10, 0) };
00349                   }
00350
00351                   Graphics xLabelsSize = new Graphics();
00352                   xLabels.Plot(xLabelsSize);
00353                   double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00354
00355                   Graphics yLabelsSize = new Graphics();
00356                   yLabels.Plot(yLabelsSize);
00357                   double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00358
00359                   ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
Baseline = TextBaselines.Top, Alignment = TextAnchors.Center };
00360                   ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00361
00362                   Plot plot = new Plot();
00363
00364                   if (vertical)
00365                   {
00366                       Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
PresentationAttributes = gridPresentationAttributes };
00367                       plot.AddPlotElement(yGrid);
00368                   }
00369                   else
00370                   {
00371                       Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
PresentationAttributes = gridPresentationAttributes };
00372                       plot.AddPlotElement(xGrid);
00373                   }
00374
00375                   plot.AddPlotElements(xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle, yTitle);
00376
00377                   double maxCount = (from el in data select el.Item2.Count).Max();
00378
```

```
00379                 for (int i = 0; i < data.Count; i++)
00380                 {
00381                     if (vertical)
00382                     {
00383                         BoxPlot box = new BoxPlot(new double[] { (i * (spacing + 1) + 0.5) * boxWidth,
      medians[i] }, new double[] { 0, 1 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00384                         {
00385                             BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00386                             WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00387                             Width = proportionalWidth ?  (boxWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (boxWidth * 0.5),
00388                             NotchSize = notches[i]
00389                         };
00390
00391                         plot.AddPlotElement(box);
00392                     }
00393                     else
00394                     {
00395                         BoxPlot box = new BoxPlot(new double[] { medians[i], (i * (spacing + 1) + 0.5)
      * boxWidth }, new double[] { 1, 0 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00396                         {
00397                             BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00398                             WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00399                             Width = proportionalWidth ?  (boxWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (boxWidth * 0.5),
00400                             NotchSize = notches[i]
00401                         };
00402
00403                         plot.AddPlotElement(box);
00404                     }
00405                 }
00406
00407                 if (showOutliers)
00408                 {
00409                     for (int i = 0; i < data.Count; i++)
00410                     {
00411                         double[][] outliers;
00412
00413                         if (vertical)
00414                         {
00415                             outliers = (from el in data[i].Item2 where el > upperWhiskers[i] || el <
      lowerWhiskers[i] select new double[] { (i * (spacing + 1) + 0.5) * boxWidth, el }).ToArray();
00416                         }
00417                         else
00418                         {
00419                             outliers = (from el in data[i].Item2 where el > upperWhiskers[i] || el <
      lowerWhiskers[i] select new double[] { el, (i * (spacing + 1) + 0.5) * boxWidth }).ToArray();
00420                         }
00421
00422                         if (outliers.Length > 0)
00423                         {
00424                             ScatterPoints<IReadOnlyList<double» outlierPoints = new
      ScatterPoints<IReadOnlyList<double»(outliers, coordinateSystem)
00425                             {
00426                                 PresentationAttributes = outlierPresentationAttributes[i %
      outlierPresentationAttributes.Count],
00427                                 Size = 3,
00428                                 DataPointElement= outlierPointElement
00429                             };
00430
00431                             plot.AddPlotElement(outlierPoints);
00432                         }
00433                     }
00434                 }
00435
00436                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00437
00438                 plot.AddPlotElement(titleLabel);
00439
00440                 return plot;
00441             }
00442
00443 /// <summary>
00444 /// Create a new box plot.
00445 /// </summary>
00446 /// <param name="data">The data to plot.</param>
00447 /// <param name="vertical">If this is <see langword="true"/> (the default), the boxes are parallel to
      the Y axis.  If this is <see langword="false"/>, the boxes are parallel to the X axis.</param>
```

```
00448 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00449 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has
      notches.  Otherwise, it does not.</param>
00450 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the boxes
      have the same width.  Otherwise, the width of each box is proportional to the number of
      samples.</param>
00451 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four
      outlier points that fall outside of the interval between the whiskers.</param>
00452 /// <param name="boxWidth">The width of the boxes in data space coordinates.</param>
00453 /// <param name="spacing">The spacing between consecutive boxes (ranging from 0 to 1).</param>
00454 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00455 /// even though the sampled values have different ranges.</param>
00456 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00457 /// even though the sampled values have different ranges.</param>
00458 /// <param name="width">The width of the plot.</param>
00459 /// <param name="height">The height of the plot.</param>
00460 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00461 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00462 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00463 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00464 /// <param name="xAxisTitle">Title for the X axis.</param>
00465 /// <param name="yAxisTitle">Title for the Y axis.</param>
00466 /// <param name="title">Title for the plot.</param>
00467 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00468 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00469 /// <param name="boxPresentationAttributes">Presentation attributes for the boxes.</param>
00470 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00471 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier
      points.</param>
00472 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00473 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00474             public static Plot BoxPlot(IReadOnlyList<IReadOnlyList<double» data, WhiskerType
      whiskerType = WhiskerType.IQR_1_5, bool useNotches = true, bool proportionalWidth = false, bool
      showOutliers = true, double boxWidth = 10, double spacing = 0.1, double?  dataRangeMin = null, double?
      dataRangeMax = null, bool vertical = true, double width = 350, double height = 250,
00475                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00476                 double axisArrowSize = 3,
00477                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00478                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00479                 string xAxisTitle = null,
00480                 string yAxisTitle = null,
00481                 string title = null,
00482                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00483                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00484                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00485                 IDataPointElement outlierPointElement = null,
00486                 IReadOnlyList<PlotElementPresentationAttributes> outlierPresentationAttributes = null,
00487                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00488             {
00489                 return BoxPlot((from el in data select ((string)null, el)).ToArray(), whiskerType,
      useNotches, proportionalWidth, showOutliers, boxWidth, spacing, dataRangeMin, dataRangeMax, vertical,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, boxPresentationAttributes, outlierPointElement,
      outlierPresentationAttributes, coordinateSystem);
00490             }
00491
00492 /// <summary>
00493 /// Create a new box plot.
00494 /// </summary>
00495 /// <param name="data">The data to plot.</param>
00496 /// <param name="vertical">If this is <see langword="true"/> (the default), the box is parallel to the
      Y axis.  If this is <see langword="false"/>, the box is parallel to the X axis.</param>
00497 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00498 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has
      notches.  Otherwise, it does not.</param>
00499 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four
      outlier points that fall outside of the interval between the whiskers.</param>
00500 /// <param name="boxWidth">The width of the box in data space coordinates.</param>
00501 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00502 /// even though the sampled values have different ranges.</param>
00503 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00504 /// even though the sampled values have different ranges.</param>
00505 /// <param name="width">The width of the plot.</param>
00506 /// <param name="height">The height of the plot.</param>
00507 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00508 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00509 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
```

```
00510 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00511 /// <param name="xAxisTitle">Title for the X axis.</param>
00512 /// <param name="yAxisTitle">Title for the Y axis.</param>
00513 /// <param name="title">Title for the plot.</param>
00514 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00515 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00516 /// <param name="boxPresentationAttributes">Presentation attributes for the box.</param>
00517 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00518 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier
      points.</param>
00519 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00520 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00521             public static Plot BoxPlot(IReadOnlyList<double> data, WhiskerType whiskerType =
      WhiskerType.IQR_1_5, bool useNotches = true, bool showOutliers = true, double boxWidth = 10, double?
      dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
      height = 250,
00522                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00523                 double axisArrowSize = 3,
00524                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00525                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00526                 string xAxisTitle = null,
00527                 string yAxisTitle = null,
00528                 string title = null,
00529                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00530                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00531                 PlotElementPresentationAttributes boxPresentationAttributes = null,
00532                 IDataPointElement outlierPointElement = null,
00533                 PlotElementPresentationAttributes outlierPresentationAttributes = null,
00534                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00535             {
00536                 return BoxPlot(new (string, IReadOnlyList<double>)[] { (null, data) }, whiskerType,
      useNotches, false, showOutliers, boxWidth, 0, dataRangeMin, dataRangeMax, vertical, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, boxPresentationAttributes == null ?  null :  new
      PlotElementPresentationAttributes[] { boxPresentationAttributes }, outlierPointElement,
      outlierPresentationAttributes == null ?  null :  new PlotElementPresentationAttributes[] {
      outlierPresentationAttributes }, coordinateSystem);
00537             }
00538         }
00539     }
00540 }
```

8.31 Plot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Collections.Immutable;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// Determines the appearance of plot elements.
00026 /// </summary>
00027     public class PlotElementPresentationAttributes
00028     {
00029 /// <summary>
00030 /// The stroke of the plot element.
00031 /// </summary>
00032         public Brush Stroke { get; set; } = Colours.Black;
00033
00034 /// <summary>
00035 /// The fill of the plot element.
00036 /// </summary>
00037         public Brush Fill { get; set; } = Colours.Black;
```

```
00038
00039 /// <summary>
00040 /// The thickness of lines in the plot element.
00041 /// </summary>
00042         public double LineWidth { get; set; } = 1;
00043
00044 /// <summary>
00045 /// The line dash style for the plot element.
00046 /// </summary>
00047         public LineDash?  LineDash { get; set; } = null;
00048
00049 /// <summary>
00050 /// The line cap for the plot element.
00051 /// </summary>
00052         public LineCaps LineCap { get; set; } = LineCaps.Square;
00053
00054 /// <summary>
00055 /// The line join for the plot element.
00056 /// </summary>
00057         public LineJoins LineJoin { get; set; } = LineJoins.Miter;
00058
00059 /// <summary>
00060 /// The font used to draw text in the plot element.
00061 /// </summary>
00062         public Font Font { get; set; } = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), 12);
00063
00064 /// <summary>
00065 /// Create a new <see cref="PlotElementPresentationAttributes"/> with the default values.
00066 /// </summary>
00067         public PlotElementPresentationAttributes()
00068         {
00069
00070         }
00071
00072 /// <summary>
00073 /// Creates a new <see cref="PlotElementPresentationAttributes"/> copying all settings from another
      instance.
00074 /// </summary>
00075 /// <param name="other">The other instance from which the settings will be copied.</param>
00076         public PlotElementPresentationAttributes(PlotElementPresentationAttributes other)
00077         {
00078             this.LineJoin = other.LineJoin;
00079             this.Stroke = other.Stroke;
00080             this.Fill = other.Fill;
00081             this.LineCap = other.LineCap;
00082             this.LineDash = other.LineDash;
00083             this.Font = other.Font;
00084             this.LineWidth = other.LineWidth;
00085         }
00086     }
00087
00088 /// <summary>
00089 /// Represents a plot element.
00090 /// </summary>
00091     public interface IPlotElement
00092     {
00093 /// <summary>
00094 /// Draw the plot element on the specified <paramref name="target"/> <see cref="Graphics"/>.
00095 /// </summary>
00096 /// <param name="target">The <see cref="Graphics"/> on which to draw.</param>
00097         void Plot(Graphics target);
00098
00099 /// <summary>
00100 /// The coordinate system used to transform the points from data space to plot space.
00101 /// </summary>
00102         ICoordinateSystem CoordinateSystem { get; }
00103     }
00104
00105 /// <summary>
00106 /// A plot element that uses an <see cref="Action"/> to draw its contents.
00107 /// </summary>
00108 /// <typeparam name="T">The type of data to plot (e.g.  <c>double[]</c>).</typeparam>
00109     public class PlotElement<T> :  IPlotElement
00110     {
00111 /// <summary>
00112 /// The action that is invoked when the plot element needs to be drawn.
00113 /// </summary>
00114         public Action<Graphics, ICoordinateSystem<T>» PlotAction { get; set; }
00115
00116 /// <summary>
00117 /// The coordinate system used to transform the points from data space to plot space.
00118 /// </summary>
00119         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00120         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00121
00122 /// <summary>
```

```
00123 /// Create a new <see cref="PlotElement{T}"/> using the specified coordinate system and plot action.
00124 /// </summary>
00125 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00126 /// <param name="plotAction">The action that is invoked when the plot element needs to be
      drawn.</param>
00127         public PlotElement(ICoordinateSystem<T> coordinateSystem, Action<Graphics,
      ICoordinateSystem<T>> plotAction)
00128         {
00129             PlotAction = plotAction;
00130             CoordinateSystem = coordinateSystem;
00131         }
00132
00133 /// <inheritdoc/>
00134         public void Plot(Graphics target)
00135         {
00136             PlotAction?.Invoke(target, CoordinateSystem);
00137         }
00138     }
00139
00140 /// <summary>
00141 /// Represents a collection of plot elements.
00142 /// </summary>
00143     public partial class Plot
00144     {
00145 /// <summary>
00146 /// The elements contained in the plot.
00147 /// </summary>
00148         public ImmutableList<IPlotElement> PlotElements { get; private set; } =
      ImmutableList.Create<IPlotElement>();
00149
00150 /// <summary>
00151 /// Add the specified plot element to the plot.
00152 /// </summary>
00153 /// <param name="plotElement">The plot element to add.</param>
00154         public void AddPlotElement(IPlotElement plotElement)
00155         {
00156             this.PlotElements = this.PlotElements.Add(plotElement);
00157         }
00158
00159 /// <summary>
00160 /// Add the specified plot elements to the plot.
00161 /// </summary>
00162 /// <param name="plotElements">The plot elements to add.</param>
00163         public void AddPlotElements(IEnumerable<IPlotElement> plotElements)
00164         {
00165             this.PlotElements = this.PlotElements.AddRange(plotElements);
00166         }
00167
00168 /// <summary>
00169 /// Add the specified plot elements to the plot.
00170 /// </summary>
00171 /// <param name="plotElements">The plot elements to add.</param>
00172         public void AddPlotElements(params IPlotElement[] plotElements)
00173         {
00174             this.AddPlotElements((IEnumerable<IPlotElement>)plotElements);
00175         }
00176
00177 /// <summary>
00178 /// Remove the specified plot element from the plot.
00179 /// </summary>
00180 /// <param name="plotElement">The plot element to remove.</param>
00181         public void RemovePlotElement(IPlotElement plotElement)
00182         {
00183             this.PlotElements = this.PlotElements.Remove(plotElement);
00184         }
00185
00186 /// <summary>
00187 /// Render the plot on the specified <paramref name="target"/> <see cref="Graphics"/>.
00188 /// </summary>
00189 /// <param name="target">The <see cref="Graphics"/> on which the plot should be drawn.</param>
00190         public void Render(Graphics target)
00191         {
00192             foreach (IPlotElement element in PlotElements)
00193             {
00194                 element.Plot(target);
00195             }
00196         }
00197
00198 /// <summary>
00199 /// Get the first <see cref="IPlotElement"/> of the specified type, or the first <see
      cref="ICoordinateSystem"/> of the specified type.
00200 /// </summary>
00201 /// <typeparam name="T">The type of plot element or coordinate system to get.</typeparam>
00202 /// <returns>The first <see cref="IPlotElement"/> of the specified type, or the first <see
      cref="ICoordinateSystem"/> of the specified type, or <see langword="null"/> if none could be
      found.</returns>
```

```
00203         public T GetFirst<T>() where T : class
00204         {
00205             foreach (IPlotElement e in this.PlotElements)
00206             {
00207                 if (e is T t)
00208                 {
00209                     return t;
00210                 }
00211                 else if (e.CoordinateSystem is T t2)
00212                 {
00213                     return t2;
00214                 }
00215             }
00216 
00217             return null;
00218         }
00219 
00220 /// <summary>
00221 /// Get all <see cref="IPlotElement"/>s of the specified type, or all <see cref="ICoordinateSystem"/>s
      of the specified type.
00222 /// </summary>
00223 /// <typeparam name="T">The type of plot element or coordinate system to get.</typeparam>
00224 /// <returns>All <see cref="IPlotElement"/>s of the specified type, or all <see
      cref="ICoordinateSystem"/>s of the specified type.</returns>
00225         public IEnumerable<T> GetAll<T>() where T : class, IPlotElement
00226         {
00227             foreach (IPlotElement e in this.PlotElements)
00228             {
00229                 if (e is T t)
00230                 {
00231                     yield return t;
00232                 }
00233                 else if (e.CoordinateSystem is T t2)
00234                 {
00235                     yield return t2;
00236                 }
00237             }
00238         }
00239 
00240 /// <summary>
00241 /// Render the plot to a suitably cropped <see cref="Page"/> object.
00242 /// </summary>
00243 /// <returns>A <see cref="Page"/> object containing the rendered plot.</returns>
00244         public Page Render()
00245         {
00246             Page pag = new Page(1, 1);
00247 
00248             this.Render(pag.Graphics);
00249 
00250             Rectangle bounds = pag.Graphics.GetBounds();
00251 
00252             pag.Crop(new Point(bounds.Location.X - bounds.Size.Width * 0.01, bounds.Location.Y -
      bounds.Size.Height * 0.01), new Size(bounds.Size.Width * 1.02, bounds.Size.Height * 1.02));
00253 
00254             return pag;
00255         }
00256 
00257 /// <summary>
00258 /// Create a new empty <see cref="Plot"/>.
00259 /// </summary>
00260         public Plot()
00261         {
00262 
00263         }
00264 
00265         private static (double minX, double minY, double maxX, double maxY, double rangeX, double
      rangeY) GetDataRange(IReadOnlyList<IReadOnlyList<double>> data)
00266         {
00267             double minX = double.MaxValue;
00268             double minY = double.MaxValue;
00269 
00270             double maxX = double.MinValue;
00271             double maxY = double.MinValue;
00272 
00273             for (int i = 0; i < data.Count; i++)
00274             {
00275                 minX = Math.Min(minX, data[i][0]);
00276                 minY = Math.Min(minY, data[i][1]);
00277 
00278                 maxX = Math.Max(maxX, data[i][0]);
00279                 maxY = Math.Max(maxY, data[i][1]);
00280             }
00281 
00282             double rangeX = maxX - minX;
00283             double rangeY = maxY - minY;
00284 
00285             return (minX, minY, maxX, maxY, rangeX, rangeY);
```

```
00286         }
00287
00288 /// <summary>
00289 /// Contains methods to create plots.
00290 /// </summary>
00291         public static partial class Create
00292         {
00293 /// <summary>
00294 /// Creates a new empty plot.
00295 /// </summary>
00296 /// <returns>An empty plot.</returns>
00297             public static Plot Empty()
00298             {
00299                 return new Plot();
00300             }
00301         }
00302     }
00303 }
```

8.32 Plot.Function1D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023     partial class Plot
00024     {
00025         partial class Create
00026         {
00027 /// <summary>
00028 /// Create a new plot for functions of one variable.
00029 /// </summary>
00030 /// <param name="functions">The functions to plot.</param>
00031 /// <param name="min">The minimum value of the function argument to plot.</param>
00032 /// <param name="max">The maximum value of the function argument to plot.</param>
00033 /// <param name="resolution">The distance between consecutive function samples.</param>
00034 /// <param name="vertical">If this is <see langword="true"/> (the default), the functions are plotted
      as y = f(x).  If this is <see langword="false"/>, the functions are plotted as x = f(y).</param>
00035 /// <param name="smooth">If this is <see langword="false"/>, consecutive function samples are joined
      by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing through
      them.</param>
00036 /// <param name="width">The width of the plot.</param>
00037 /// <param name="height">The height of the plot.</param>
00038 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00039 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00040 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00041 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00042 /// <param name="xAxisTitle">Title for the X axis.</param>
00043 /// <param name="yAxisTitle">Title for the Y axis.</param>
00044 /// <param name="title">Title for the plot.</param>
00045 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00046 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00047 /// <param name="linePresentationAttributes">Presentation attributes for the function line.</param>
00048 /// <param name="pointPresentationAttributes">Presentation attributes for the symbols drawn at the
      sampled function points.</param>
00049 /// <param name="pointSize">Size of the symbols drawn at the sampled function points.</param>
00050 /// <param name="dataPointElements">Symbols drawn at the sampled function points.</param>
00051 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00052 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00053             public static Plot Function(IReadOnlyList<Func<double, double>> functions, double min,
      double max, double resolution = double.NaN, bool vertical = true, bool smooth = false, double width =
      350, double height = 250,
00054                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00055                 double axisArrowSize = 3,
```

```
00056                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00057                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00058                 string xAxisTitle = null,
00059                 string yAxisTitle = null,
00060                 string title = null,
00061                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00062                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00063                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00064                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00065                 IReadOnlyList<double> pointSize = null,
00066                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00067                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00068             {
00069                 if (double.IsNaN(resolution))
00070                 {
00071                     if (coordinateSystem == null)
00072                     {
00073                         resolution = (max - min) / 500;
00074                     }
00075                     else
00076                     {
00077                         resolution = coordinateSystem.Resolution[0];
00078                     }
00079                 }
00080 
00081                 int steps = (int)Math.Ceiling((max - min) / resolution);
00082 
00083                 List<double[]>[] data = new List<double[]>[functions.Count];
00084 
00085                 for (int i = 0; i < functions.Count; i++)
00086                 {
00087                     data[i] = new List<double[]>(steps + 1);
00088 
00089                     for (int j = 0; j <= steps; j++)
00090                     {
00091                         double x = min + (max - min) / steps * j;
00092                         double y = functions[i](x);
00093 
00094                         if (!double.IsNaN(y))
00095                         {
00096                             if (vertical)
00097                             {
00098                                 data[i].Add(new double[] { x, y });
00099                             }
00100                             else
00101                             {
00102                                 data[i].Add(new double[] { y, x });
00103                             }
00104                         }
00105                     }
00106                 }
00107 
00108                 return Create.LineCharts(data, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSize, dataPointElements,
      coordinateSystem);
00109             }
00110 
00111 /// <summary>
00112 /// Create a new plot for a function of one variable.
00113 /// </summary>
00114 /// <param name="function">The function to plot.</param>
00115 /// <param name="min">The minimum value of the function argument to plot.</param>
00116 /// <param name="max">The maximum value of the function argument to plot.</param>
00117 /// <param name="resolution">The distance between consecutive function samples.</param>
00118 /// <param name="vertical">If this is <see langword="true"/> (the default), the function is plotted as
      y = f(x).  If this is <see langword="false"/>, the function is plotted as x = f(y).</param>
00119 /// <param name="smooth">If this is <see langword="false"/>, consecutive function samples are joined
      by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing through
      them.</param>
00120 /// <param name="width">The width of the plot.</param>
00121 /// <param name="height">The height of the plot.</param>
00122 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00123 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00124 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00125 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00126 /// <param name="xAxisTitle">Title for the X axis.</param>
00127 /// <param name="yAxisTitle">Title for the Y axis.</param>
00128 /// <param name="title">Title for the plot.</param>
00129 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00130 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00131 /// <param name="linePresentationAttributes">Presentation attributes for the function line.</param>
00132 /// <param name="pointPresentationAttributes">Presentation attributes for the symbols drawn at the
      sampled function points.</param>
00133 /// <param name="pointSize">Size of the symbols drawn at the sampled function points.</param>
00134 /// <param name="dataPointElement">Symbol drawn at the sampled function points.</param>
```

```
00135 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00136 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00137             public static Plot Function(Func<double, double> function, double min, double max, double
      resolution = double.NaN, bool vertical = true, bool smooth = false, double width = 350, double height
      = 250,
00138                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00139                 double axisArrowSize = 3,
00140                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00141                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00142                 string xAxisTitle = null,
00143                 string yAxisTitle = null,
00144                 string title = null,
00145                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00146                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00147                 PlotElementPresentationAttributes linePresentationAttributes = null,
00148                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00149                 double pointSize = 0,
00150                 IDataPointElement dataPointElement = null,
00151                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00152             {
00153                 return Function(new Func<double, double>[] { function }, min, max, resolution,
      vertical, smooth, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, linePresentationAttributes == null ?  null :
      new PlotElementPresentationAttributes[] { linePresentationAttributes }, pointPresentationAttributes ==
      null ?  null :  new PlotElementPresentationAttributes[] { pointPresentationAttributes }, new double[]
      { pointSize }, dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement },
      coordinateSystem);
00154             }
00155         }
00156     }
00157 }
```

8.33 Plot.Function2D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023     partial class Plot
00024     {
00025         partial class Create
00026         {
00027 /// <summary>
00028 /// Create a new plot for a function of two variables.
00029 /// </summary>
00030 /// <param name="function">The function grid to plot.</param>
00031 /// <param name="plotType">The type of plot to produce.</param>
00032 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00033 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
```

```
00045 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00048             public static Plot Function(Function2DGrid function, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 IDataPointElement pointElement = null,
00058                 Func<double, Colour> colouring = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061
00062                 if (axisPresentationAttributes == null)
00063                 {
00064                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00065                 }
00066
00067                 if (axisLabelPresentationAttributes == null)
00068                 {
00069                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00070                 }
00071
00072                 if (axisTitlePresentationAttributes == null)
00073                 {
00074                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00075                 }
00076
00077                 if (pointElement == null)
00078                 {
00079                     pointElement = new PathDataPointElement();
00080                 }
00081
00082                 if (coordinateSystem == null)
00083                 {
00084                     coordinateSystem = new LinearCoordinateSystem2D(function.DataPoints, width,
      height);
00085                 }
00086
00087                 if (titlePresentationAttributes == null)
00088                 {
00089                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00090                 }
00091
00092                 if (colouring == null)
00093                 {
00094                     colouring = Gradients.Viridis;
00095                 }
00096
00097                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(function.DataPoints);
00098
00099                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00100                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00101                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00102                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00103
00104                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00105                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00106                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00107                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00108
00109                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
```

```
00110                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00111                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00112                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00113
00114
00115                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00116                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00117                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00118                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00119
00120                 //Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00121                 //Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00122
00123                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00124                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00125
00126                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00127                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00128
00129                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00130                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00131
00132                 Graphics xLabelsSize = new Graphics();
00133                 xLabels.Plot(xLabelsSize);
00134                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00135
00136                 Graphics yLabelsSize = new Graphics();
00137                 yLabels.Plot(yLabelsSize);
00138                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00139
00140                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00141                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00142
00143                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00144
00145                 Function2D functionPlot = new Function2D(function, coordinateSystem)
00146                 {
00147                     RasterResolutionX = rasterResolutionX,
00148                     RasterResolutionY = rasterResolutionY,
00149                     SampledPointElement = pointElement,
00150                     Type = plotType,
00151                     Colouring = colouring
00152                 };
00153
00154                 Plot tbr = new Plot();
00155                 tbr.AddPlotElements(functionPlot, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00156
00157                 tbr.AddPlotElement(titleLabel);
00158
00159                 return tbr;
00160             }
00161
00162 /// <summary>
00163 /// Create a new plot for a function of two variables.
00164 /// </summary>
```

```
00165 /// <param name="function">The function to plot.</param>
00166 /// <param name="minX">The minimum value of the first function argument to plot.</param>
00167 /// <param name="minY">The minimum value of the second function argument to plot.</param>
00168 /// <param name="maxX">The maximum value of the first function argument to plot.</param>
00169 /// <param name="maxY">The maximum value of the second function argument to plot.</param>
00170 /// <param name="resolutionX">The distance between consecutive function samples on the X axis.</param>
00171 /// <param name="resolutionY">The distance between consecutive function samples on the Y axis.</param>
00172 /// <param name="gridType">The type of grid along which to sample the function.  If it is <see
      langword="null" />, this is determined automatically.</param>
00173 /// <param name="plotType">The type of plot to produce.</param>
00174 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00175 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00176 /// <param name="width">The width of the plot.</param>
00177 /// <param name="height">The height of the plot.</param>
00178 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00179 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00180 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00181 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00182 /// <param name="xAxisTitle">Title for the X axis.</param>
00183 /// <param name="yAxisTitle">Title for the Y axis.</param>
00184 /// <param name="title">Title for the plot.</param>
00185 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00186 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
00187 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00188 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00189 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00190             public static Plot Function(Func<double[], double> function, double minX, double minY,
      double maxX, double maxY, double resolutionX = double.NaN, double resolutionY = double.NaN,
      Function2DGrid.GridType?  gridType = null, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00191                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00192                 double axisArrowSize = 3,
00193                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00194                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00195                 string xAxisTitle = null,
00196                 string yAxisTitle = null,
00197                 string title = null,
00198                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00199                 IDataPointElement pointElement = null,
00200                 Func<double, Colour> colouring = null,
00201                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00202             {
00203                 if (double.IsNaN(resolutionX) && double.IsNaN(resolutionY))
00204                 {
00205                     double ratio = width / height;
00206 
00207                     double stepsY = Math.Sqrt(4096 / ratio);
00208                     double stepsX = ratio * stepsY;
00209 
00210                     resolutionX = (maxX - minX) / stepsX;
00211                     resolutionY = (maxY - minY) / stepsY;
00212                 }
00213                 else if (double.IsNaN(resolutionX) && !double.IsNaN(resolutionY))
00214                 {
00215                     double ratio = width / height;
00216 
00217                     double stepsY = (maxY - minY) / resolutionY;
00218                     double stepsX = ratio * stepsY;
00219 
00220                     resolutionX = (maxX - minX) / stepsX;
00221                 }
00222                 else if (!double.IsNaN(resolutionX) && double.IsNaN(resolutionY))
00223                 {
00224                     double ratio = width / height;
00225 
00226                     double stepsX = (maxX - minX) / resolutionX;
00227                     double stepsY = stepsX / ratio;
00228 
00229                     resolutionY = (maxY - minY) / stepsY;
00230                 }
00231 
00232                 int stepsCountX = (int)Math.Floor((maxX - minX) / resolutionX);
00233                 int stepsCountY = (int)Math.Floor((maxY - minY) / resolutionY);
00234 
00235                 if (gridType == null)
00236                 {
00237                     switch (plotType)
00238                     {
00239                         case Function2D.PlotType.SampledPoints:
00240                             gridType = Function2DGrid.GridType.HexagonHorizontal;
```

```
00241                             break;
00242
00243                         case Function2D.PlotType.Tessellation:
00244                             if ((stepsCountX + 1) * (stepsCountY + 1) <= 4096)
00245                             {
00246                                 gridType = Function2DGrid.GridType.Irregular;
00247                             }
00248                             else
00249                             {
00250                                 gridType = Function2DGrid.GridType.HexagonHorizontal;
00251                             }
00252                             break;
00253
00254                         case Function2D.PlotType.Raster:
00255                             gridType = Function2DGrid.GridType.Rectangular;
00256                             break;
00257                     }
00258                 }
00259
00260                 Function2DGrid grid = new Function2DGrid(function, minX, minY, maxX, maxY,
      stepsCountX, stepsCountY, gridType.Value);
00261
00262                 return Function(grid, plotType, rasterResolutionX, rasterResolutionY, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      pointElement, colouring, coordinateSystem);
00263             }
00264
00265 /// <summary>
00266 /// Create a new plot for a function of two variables.
00267 /// </summary>
00268 /// <param name="function">The function to plot.</param>
00269 /// <param name="minX">The minimum value of the first function argument to plot.</param>
00270 /// <param name="minY">The minimum value of the second function argument to plot.</param>
00271 /// <param name="maxX">The maximum value of the first function argument to plot.</param>
00272 /// <param name="maxY">The maximum value of the second function argument to plot.</param>
00273 /// <param name="resolutionX">The distance between consecutive function samples on the X axis.</param>
00274 /// <param name="resolutionY">The distance between consecutive function samples on the Y axis.</param>
00275 /// <param name="gridType">The type of grid along which to sample the function.  If it is <see
      langword="null" />, this is determined automatically.</param>
00276 /// <param name="plotType">The type of plot to produce.</param>
00277 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00278 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00279 /// <param name="width">The width of the plot.</param>
00280 /// <param name="height">The height of the plot.</param>
00281 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00282 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00283 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00284 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00285 /// <param name="xAxisTitle">Title for the X axis.</param>
00286 /// <param name="yAxisTitle">Title for the Y axis.</param>
00287 /// <param name="title">Title for the plot.</param>
00288 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00289 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
00290 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00291 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00292 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00293             public static Plot Function(Func<double, double, double> function, double minX, double
      minY, double maxX, double maxY, double resolutionX = double.NaN, double resolutionY = double.NaN,
      Function2DGrid.GridType?  gridType = null, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00294                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00295                 double axisArrowSize = 3,
00296                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00297                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00298                 string xAxisTitle = null,
00299                 string yAxisTitle = null,
00300                 string title = null,
00301                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00302                 IDataPointElement pointElement = null,
00303                 Func<double, Colour> colouring = null,
00304                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00305             {
00306                 return Function(x => function(x[0], x[1]), minX, minY, maxX, maxY, resolutionX,
      resolutionY, gridType, plotType, rasterResolutionX, rasterResolutionY, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      pointElement, colouring, coordinateSystem);
00307             }
00308         }
```

```
00309     }
00310 }
```

8.34 Plot.Histogram.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026 /// <summary>
00027 /// Describes the kind of normalisation that is performed.
00028 /// </summary>
00029         public enum NormalisationMode
00030         {
00031 /// <summary>
00032 /// No normalisation is performed.
00033 /// </summary>
00034             None,
00035
00036 /// <summary>
00037 /// Values are normalised so that the maximum of each distribution corresponds to the same value.
00038 /// </summary>
00039             Maximum,
00040
00041 /// <summary>
00042 /// Values are normalised so that the area covered by each distribution is the same.
00043 /// </summary>
00044             Area
00045         }
00046
00047         internal static (double q1, double q3, double iqr) IQR(IEnumerable<double> values)
00048         {
00049             double[] values2 = (from el in values orderby el ascending select el).ToArray();
00050
00051             double q1, q3;
00052
00053             if (values2.Length < 4)
00054             {
00055                 q1 = values2[0];
00056                 q3 = values2[values2.Length - 1];
00057             }
00058             else if (values2.Length == 4)
00059             {
00060                 q1 = (values2[0] + values2[1]) * 0.5;
00061                 q3 = (values2[2] + values2[3]) * 0.5;
00062             }
00063             else if (values2.Length % 2 == 0)
00064             {
00065                 q1 = values2.Length % 4 == 0 ?  (values2[values2.Length / 4] + values2[values2.Length
      / 4 + 1]) * 0.5 :  values2[values2.Length / 4];
00066                 q3 = values2.Length % 4 == 0 ?  (values2[3 * values2.Length / 4] + values2[3 *
      values2.Length / 4 + 1]) * 0.5 :  values2[3 * values2.Length / 4];
00067             }
00068             else if (values2.Length % 4 == 1)
00069             {
00070                 int n = (values2.Length - 1) / 4;
00071                 q1 = 0.75 * values2[n - 1] + 0.25 * values2[n];
00072                 q3 = 0.25 * values2[3 * n] + 0.75 * values2[3 * n + 1];
00073             }
00074             else
00075             {
00076                 int n = (values2.Length - 3) / 4;
```

```
00077                 q1 = 0.75 * values2[n] + 0.25 * values2[n + 1];
00078                 q3 = 0.25 * values2[3 * n + 1] + 0.75 * values2[3 * n + 2];
00079             }
00080 
00081             return (q1, q3, q3 - q1);
00082         }
00083 
00084         partial class Create
00085         {
00086             private static Plot Histogram(IReadOnlyList<IReadOnlyList<double» data, double
      overrideMax, double overrideMin, bool vertical = true, double margin = 0, double width = 350, double
      height = 250,
00087                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00088                 double axisArrowSize = 3,
00089                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00090                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00091                 string xAxisTitle = null,
00092                 string yAxisTitle = null,
00093                 string title = null,
00094                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00095                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00096                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00097                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00098             {
00099                 if (axisPresentationAttributes == null)
00100                 {
00101                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00102                 }
00103 
00104                 if (gridPresentationAttributes == null)
00105                 {
00106                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00107                 }
00108 
00109                 if (axisLabelPresentationAttributes == null)
00110                 {
00111                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00112                 }
00113 
00114                 if (axisTitlePresentationAttributes == null)
00115                 {
00116                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00117                 }
00118 
00119                 if (dataPresentationAttributes == null)
00120                 {
00121                     dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) };
00122                 }
00123 
00124                 double baselineValue = 0;
00125 
00126                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00127                 {
00128                     baselineValue = 1;
00129                 }
00130 
00131                 Bars bars;
00132 
00133                 if (baselineValue == 0)
00134                 {
00135                     bars = new Bars(data, coordinateSystem, vertical) { PresentationAttributes =
      dataPresentationAttributes, Margin = margin };
00136                 }
00137                 else
00138                 {
00139                     bars = new Bars(data, new Comparison<IReadOnlyList<double»(vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x, y) =>
      Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double»(pt => new
      double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] }, coordinateSystem) {
      PresentationAttributes = dataPresentationAttributes, Margin = margin };
00140                 }
00141 
00142                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(data);
00143 
00144                 double dataMinX = minX;
00145                 double dataMinY = minY;
00146                 double dataMaxX = maxX;
00147                 double dataMaxY = maxY;
00148 
```

```
00149                 if (coordinateSystem == null)
00150                 {
00151                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
00152 
00153                     if (vertical)
00154                     {
00155                         minY = Math.Min(minY, 0);
00156                         maxY = Math.Max(maxY, 0);
00157 
00158                         if (!double.IsNaN(overrideMax))
00159                         {
00160                             maxY = overrideMax;
00161                         }
00162 
00163                         if (!double.IsNaN(overrideMin))
00164                         {
00165                             minY = overrideMin;
00166                         }
00167 
00168                         rangeY = maxY - minY;
00169 
00170                         linCoords.MinY = minY - rangeY * 0.1;
00171                         linCoords.MaxY = maxY + rangeY * 0.1;
00172 
00173 
00174                         if (bars.Data.Count >= 2)
00175                         {
00176                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00177                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00178 
00179                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00180                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00181 
00182                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00183                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00184 
00185                             rangeX = maxX - minX;
00186                             linCoords.MinX = minX;
00187                             linCoords.MaxX = maxX;
00188                         }
00189                     }
00190                     else
00191                     {
00192                         minX = Math.Min(minX, 0);
00193                         maxX = Math.Max(maxX, 0);
00194 
00195                         if (!double.IsNaN(overrideMax))
00196                         {
00197                             maxX = overrideMax;
00198                         }
00199 
00200                         if (!double.IsNaN(overrideMin))
00201                         {
00202                             minX = overrideMin;
00203                         }
00204 
00205                         rangeX = maxX - minX;
00206 
00207                         linCoords.MinX = minX - rangeX * 0.1;
00208                         linCoords.MaxX = maxX + rangeX * 0.1;
00209 
00210 
00211                         if (bars.Data.Count >= 2)
00212                         {
00213                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00214                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00215 
00216                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00217                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00218 
00219                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00220                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00221 
00222                             rangeY = maxY - minY;
00223                             linCoords.MinY = minY;
00224                             linCoords.MaxY = maxY;
00225                         }
00226                     }
00227 
00228                     coordinateSystem = linCoords;
00229                     bars.CoordinateSystem = coordinateSystem;
00230                 }
00231                 else
00232                 {
00233                     if (vertical)
00234                     {
```

```
00235                         minY = Math.Min(minY, baselineValue);
00236                         rangeY = maxY - minY;
00237 
00238                         if (bars.Data.Count >= 2)
00239                         {
00240                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00241                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00242 
00243                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00244                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00245 
00246                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00247                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00248 
00249                             rangeX = maxX - minX;
00250                         }
00251                     }
00252                     else
00253                     {
00254                         minX = Math.Min(minX, baselineValue);
00255                         rangeX = maxX - minX;
00256 
00257                         if (bars.Data.Count >= 2)
00258                         {
00259                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00260                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00261 
00262                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00263                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00264 
00265                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00266                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00267 
00268                             rangeY = maxY - minY;
00269                         }
00270                     }
00271                 }
00272 
00273 
00274 
00275                 if (titlePresentationAttributes == null)
00276                 {
00277                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00278                 }
00279 
00280                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00281                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00282                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00283                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00284 
00285                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00286                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00287                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00288                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00289 
00290                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00291                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00292                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00293                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00294 
00295 
00296                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00297                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00298                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
```

```
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00299                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00300
00301                 if (vertical)
00302                 {
00303                     Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00304                     Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
00305
00306                     Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00307                     Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00308
00309                     Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00310                     Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
00311
00312                     Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00313                     Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
00314
00315                     p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
00316                     p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
00317                     p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
00318                     p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
00319                 }
00320                 else
00321                 {
00322                     Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00323                     Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
00324
00325                     Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00326                     Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
00327
00328                     Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00329                     Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00330
00331                     Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00332                     Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
      });
00333
00334                     p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
00335                     p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
00336                     p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
00337                     p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
00338                 }
00339
00340                 Grid grid;
00341
00342                 if (vertical)
00343                 {
00344                     grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00345                 }
00346                 else
00347                 {
00348                     grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00349                 }
00350
00351                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00352                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00353
00354                 int every = (int)Math.Ceiling((double)data.Count / 5);
00355                 int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
00356
00357                 ContinuousAxisTicks xTicks;
00358                 ContinuousAxisTicks yTicks;
```

```
00359
00360                 if (vertical)
00361                 {
00362                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00363                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00364                 }
00365                 else
00366                 {
00367                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00368                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00369                 }
00370
00371                 ContinuousAxisLabels xLabels;
00372                 ContinuousAxisLabels yLabels;
00373
00374                 if (vertical)
00375                 {
00376                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
00377                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00378
00379                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      xLabels.TextFormat;
00380                     xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00381                 }
00382                 else
00383                 {
00384                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00385                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
00386
00387                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      yLabels.TextFormat;
00388                     yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00389                 }
00390
00391                 Graphics xLabelsSize = new Graphics();
00392                 xLabels.Plot(xLabelsSize);
00393                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00394
00395                 Graphics yLabelsSize = new Graphics();
00396                 yLabels.Plot(yLabelsSize);
00397                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00398
00399                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00400                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00401
00402                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00403
00404                 Plot tbr = new Plot();
00405                 tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
      yTitle, bars, titleLabel);
00406
00407                 return tbr;
00408             }
00409
00410 /// <summary>
00411 /// Create a new histogram.
00412 /// </summary>
00413 /// <param name="data">The data to plot.</param>
00414 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00415 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00416 /// <param name="width">The width of the plot.</param>
```

```
00417 /// <param name="height">The height of the plot.</param>
00418 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00419 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00420 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00421 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00422 /// <param name="xAxisTitle">Title for the X axis.</param>
00423 /// <param name="yAxisTitle">Title for the Y axis.</param>
00424 /// <param name="title">Title for the plot.</param>
00425 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00426 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00427 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00428 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00429 /// <returns>A <see cref="Plot"/> containing the histogram.</returns>
00430             public static Plot Histogram(IReadOnlyList<IReadOnlyList<double» data, bool vertical =
      true, double margin = 0, double width = 350, double height = 250,
00431                     PlotElementPresentationAttributes axisPresentationAttributes = null,
00432                     double axisArrowSize = 3,
00433                     PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00434                     PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00435                     string xAxisTitle = null,
00436                     string yAxisTitle = null,
00437                     string title = null,
00438                     PlotElementPresentationAttributes titlePresentationAttributes = null,
00439                     PlotElementPresentationAttributes gridPresentationAttributes = null,
00440                     PlotElementPresentationAttributes dataPresentationAttributes = null,
00441                     IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00442             {
00443                 return Histogram(data, double.NaN, double.NaN, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00444             }
00445
00446 /// <summary>
00447 /// Create a new histogram.
00448 /// </summary>
00449 /// <param name="data">The data whose distribution will be plotted.</param>
00450 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00451 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00452 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
      determined automatically using the Freedman-Diaconis rule.</param>
00453 /// <param name="underflow">Values smaller than this will be excluded from the main plot and moved
      into an underflow bin.</param>
00454 /// <param name="overflow">Values larger than this will be excluded from the main plot and moved into
      an overflow bin.</param>
00455 /// <param name="width">The width of the plot.</param>
00456 /// <param name="height">The height of the plot.</param>
00457 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00458 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00459 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00460 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00461 /// <param name="xAxisTitle">Title for the X axis.</param>
00462 /// <param name="yAxisTitle">Title for the Y axis.</param>
00463 /// <param name="title">Title for the plot.</param>
00464 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00465 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00466 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00467 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00468 /// <returns>A <see cref="Plot"/> containing the histogram.</returns>
00469             public static Plot Histogram(IReadOnlyList<double> data, bool vertical = true, double
      margin = 0, int binCount = -1, double underflow = double.NegativeInfinity, double overflow =
      double.PositiveInfinity, double width = 350, double height = 250,
00470                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00471                 double axisArrowSize = 3,
00472                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00473                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00474                 string xAxisTitle = null,
00475                 string yAxisTitle = null,
00476                 string title = null,
00477                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00478                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00479                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00480                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00481             {
00482                 if (dataPresentationAttributes == null)
00483                 {
00484                     dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 0.5, Fill = new
      SolidColourBrush(Colour.FromRgb(0, 114, 178)) };
00485                 }
00486
00487                 int underflowCount = 0;
```

```
00488                 int overflowCount = 0;
00489
00490                 List<double> binnableData = new List<double>(data.Count);
00491
00492                 double min = double.MaxValue;
00493                 double max = double.MinValue;
00494
00495                 for (int i = 0; i < data.Count; i++)
00496                 {
00497                     if (data[i] > underflow && data[i] < overflow)
00498                     {
00499                         min = Math.Min(min, data[i]);
00500                         max = Math.Max(max, data[i]);
00501                         binnableData.Add(data[i]);
00502                     }
00503                     else if (data[i] <= underflow)
00504                     {
00505                         underflowCount++;
00506                     }
00507                     else if (data[i] >= overflow)
00508                     {
00509                         overflowCount++;
00510                     }
00511                 }
00512
00513                 if (binCount <= 1)
00514                 {
00515                     (double _, double _, double iqr) = IQR(binnableData);
00516                     double h2 = 2 * iqr / Math.Pow(binnableData.Count, 1.0 / 3.0);
00517
00518                     if (h2 > 0)
00519                     {
00520                         binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
00521                     }
00522                     else
00523                     {
00524                         binCount = 2;
00525                     }
00526                 }
00527
00528                 int[] bins = new int[binCount];
00529
00530                 if (max > min)
00531                 {
00532                     for (int i = 0; i < binnableData.Count; i++)
00533                     {
00534                         int index = (int)Math.Min(binCount - 1, Math.Floor((binnableData[i] - min) /
      (max - min) * binCount));
00535
00536                         bins[index]++;
00537                     }
00538                 }
00539                 else
00540                 {
00541                     bins[0] = binnableData.Count;
00542                 }
00543
00544                 (string, double)[] binnedData = new (string, double)[bins.Length];
00545
00546                 for (int i = 0; i < bins.Length; i++)
00547                 {
00548                     double binStart = min + (max - min) / binCount * i;
00549                     double binEnd = min + (max - min) / binCount * (i + 1);
00550
00551                     string formatString;
00552
00553                     if (binEnd - binStart >= 10)
00554                     {
00555                         formatString = "0";
00556                     }
00557                     else if (binEnd - binStart >= 1)
00558                     {
00559                         formatString = "0.0";
00560                     }
00561                     else if (binEnd == binStart)
00562                     {
00563                         formatString = "0." + new string('0',
      -(int)Math.Floor(Math.Log10(Math.Abs(binEnd))) + 1);
00564                     }
00565                     else
00566                     {
00567                         formatString = "0." + new string('0', -(int)Math.Floor(Math.Log10(binEnd -
      binStart)) + 1);
00568                     }
00569
00570                     string binLabel = "[" + binStart.ToString(formatString,
      System.Globalization.CultureInfo.InvariantCulture) + ", " + binEnd.ToString(formatString,
```

```
    System.Globalization.CultureInfo.InvariantCulture);
00571
00572                   if (i < bins.Length - 1)
00573                   {
00574                       binLabel += ")";
00575                   }
00576                   else
00577                   {
00578                       binLabel += "]";
00579                   }
00580
00581                   binnedData[i] = (binLabel, bins[i]);
00582               }
00583
00584               double maxY = Math.Max(Math.Max(underflowCount, overflowCount), bins.Max());
00585
00586               Plot plot = Create.BarChart(binnedData, maxY, double.NaN, vertical, margin, width,
    height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
    axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
    gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00587
00588               if (vertical)
00589               {
00590                   DataLabels<IReadOnlyList<double» xLabels =
    plot.GetFirst<DataLabels<IReadOnlyList<double»>();
00591
00592                   xLabels.Alignment = TextAnchors.Left;
00593
00594                   xLabels.Rotation = (a, b) => Math.PI / 6;
00595
00596                   xLabels.Baseline = TextBaselines.Middle;
00597
00598                   Func<int, IReadOnlyList<double>, object> prevLabels = xLabels.Label;
00599
00600                   xLabels.Label = (i, x) =>
00601                   {
00602                       if (i == 0 || i == bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i
    == Math.Floor(bins.Length * 0.25) || i == Math.Floor(bins.Length * 0.75))
00603                       {
00604                           return prevLabels(i, x);
00605                       }
00606                       else
00607                       {
00608                           return null;
00609                       }
00610                   };
00611
00612                   plot.GetFirst<ContinuousAxisTicks>().SizeAbove = i => (i == 0 || i == bins.Length
    - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00613                   plot.GetFirst<ContinuousAxisTicks>().SizeBelow = i => (i == 0 || i == bins.Length
    - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00614
00615                   Graphics xLabelsSize = new Graphics();
00616                   xLabels.Plot(xLabelsSize);
00617                   double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00618
00619                   plot.GetFirst<ContinuousAxisTitle>().Position = xLabelsHeight + 20;
00620
00621               }
00622               else
00623               {
00624                   DataLabels<IReadOnlyList<double» xLabels =
    plot.GetFirst<DataLabels<IReadOnlyList<double»>();
00625
00626                   Func<int, IReadOnlyList<double>, object> prevLabels = xLabels.Label;
00627
00628                   xLabels.Label = (i, x) =>
00629                   {
00630                       if (i == 0 || i == bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i
    == Math.Floor(bins.Length * 0.25) || i == Math.Floor(bins.Length * 0.75))
00631                       {
00632                           return prevLabels(i, x);
00633                       }
00634                       else
00635                       {
00636                           return null;
00637                       }
00638                   };
00639
00640                   plot.GetAll<ContinuousAxisTicks>().ElementAt(1).SizeAbove = i => (i == 0 || i ==
    bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00641                   plot.GetAll<ContinuousAxisTicks>().ElementAt(1).SizeBelow = i => (i == 0 || i ==
    bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
```

```
00642                 }
00643
00644                 if (underflowCount > 0)
00645                 {
00646                     if (vertical)
00647                     {
00648                         ContinuousAxis xAxis = plot.GetFirst<ContinuousAxis>();
00649                         xAxis.StartPoint = new double[] { xAxis.StartPoint[0] - 1.5,
      xAxis.StartPoint[1] };
00650
00651                         ContinuousAxis yAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00652
00653                         yAxis.StartPoint = new double[] { yAxis.StartPoint[0] - 1.5,
      yAxis.StartPoint[1] };
00654                         yAxis.EndPoint = new double[] { yAxis.EndPoint[0] - 1.5, yAxis.EndPoint[1] };
00655
00656                         Grid grid = plot.GetFirst<Grid>();
00657                         grid.Side1Start = new double[] { grid.Side1Start[0] - 1.5, grid.Side1Start[1]
      };
00658                         grid.Side1End = new double[] { grid.Side1End[0] - 1.5, grid.Side1End[1] };
00659
00660                         ContinuousAxisTicks yTicks = plot.GetAll<ContinuousAxisTicks>().ElementAt(1);
00661                         yTicks.StartPoint = new double[] { yTicks.StartPoint[0] - 1.5,
      yTicks.StartPoint[1] };
00662                         yTicks.EndPoint = new double[] { yTicks.EndPoint[0] - 1.5, yTicks.EndPoint[1]
      };
00663
00664                         ContinuousAxisLabels yLabels = plot.GetFirst<ContinuousAxisLabels>();
00665                         yLabels.StartPoint = new double[] { yLabels.StartPoint[0] - 1.5,
      yLabels.StartPoint[1] };
00666                         yLabels.EndPoint = new double[] { yLabels.EndPoint[0] - 1.5,
      yLabels.EndPoint[1] };
00667
00668                         ContinuousAxisTitle yTitle = plot.GetAll<ContinuousAxisTitle>().ElementAt(1);
00669                         yTitle.StartPoint = yAxis.StartPoint;
00670                         yTitle.EndPoint = yAxis.EndPoint;
00671
00672                     }
00673                     else
00674                     {
00675                         ContinuousAxis xAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00676                         xAxis.StartPoint = new double[] { xAxis.StartPoint[0], xAxis.StartPoint[1] -
      1.5 };
00677
00678                         ContinuousAxis yAxis = plot.GetFirst<ContinuousAxis>();
00679
00680                         yAxis.StartPoint = new double[] { yAxis.StartPoint[0], yAxis.StartPoint[1] -
      1.5 };
00681                         yAxis.EndPoint = new double[] { yAxis.EndPoint[0], yAxis.EndPoint[1] - 1.5 };
00682
00683                         Grid grid = plot.GetFirst<Grid>();
00684                         grid.Side2Start = new double[] { grid.Side2Start[0], grid.Side2Start[1] - 1.5
      };
00685                         grid.Side2End = new double[] { grid.Side2End[0], grid.Side2End[1] - 1.5 };
00686
00687                         ContinuousAxisTicks yTicks = plot.GetFirst<ContinuousAxisTicks>();
00688                         yTicks.StartPoint = new double[] { yTicks.StartPoint[0], yTicks.StartPoint[1]
      - 1.5 };
00689                         yTicks.EndPoint = new double[] { yTicks.EndPoint[0], yTicks.EndPoint[1] - 1.5
      };
00690
00691                         ContinuousAxisLabels yLabels = plot.GetFirst<ContinuousAxisLabels>();
00692                         yLabels.StartPoint = new double[] { yLabels.StartPoint[0],
      yLabels.StartPoint[1] - 1.5 };
00693                         yLabels.EndPoint = new double[] { yLabels.EndPoint[0], yLabels.EndPoint[1] -
      1.5 };
00694
00695                         ContinuousAxisTitle yTitle = plot.GetFirst<ContinuousAxisTitle>();
00696                         yTitle.StartPoint = yAxis.StartPoint;
00697                         yTitle.EndPoint = yAxis.EndPoint;
00698                     }
00699                 }
00700
00701                 if (overflowCount > 0)
00702                 {
00703                     if (vertical)
00704                     {
00705                         ContinuousAxis xAxis = plot.GetFirst<ContinuousAxis>();
00706                         xAxis.EndPoint = new double[] { xAxis.EndPoint[0] + 1.5, xAxis.EndPoint[1] };
00707
00708                         Grid grid = plot.GetFirst<Grid>();
00709                         grid.Side2Start = new double[] { grid.Side2Start[0] + 1.5, grid.Side2Start[1]
      };
00710                         grid.Side2End = new double[] { grid.Side2End[0] + 1.5, grid.Side2End[1] };
00711                     }
00712                     else
00713                     {
```

```
00714                           ContinuousAxis xAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00715                           xAxis.EndPoint = new double[] { xAxis.EndPoint[0], xAxis.EndPoint[1] + 1.5 };
00716 
00717                           Grid grid = plot.GetFirst<Grid>();
00718                           grid.Side1Start = new double[] { grid.Side1Start[0], grid.Side1Start[1] + 1.5
      };
00719                           grid.Side1End = new double[] { grid.Side1End[0], grid.Side1End[1] + 1.5 };
00720 
00721                           TextLabel<IReadOnlyList<double> titleLabel =
      plot.GetFirst<TextLabel<IReadOnlyList<double>>();
00722                           titleLabel.Position = new double[] { titleLabel.Position[0],
      titleLabel.Position[1] + 1.5 };
00723                       }
00724                   }
00725 
00726                   if (overflowCount > 0 || underflowCount > 0)
00727                   {
00728                       DataLabels<IReadOnlyList<double> xLabels =
      plot.GetFirst<DataLabels<IReadOnlyList<double>>();
00729 
00730                       List<IReadOnlyList<double> xLabelData = xLabels.Data.ToList();
00731 
00732                       List<IReadOnlyList<double> barData = new List<IReadOnlyList<double>();
00733 
00734                       if (underflowCount > 0)
00735                       {
00736                           if (vertical)
00737                           {
00738                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { -1.5, 0
      }));
00739 
00740                               barData.Add(new double[] { -1.5, underflowCount });
00741                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { -0.5, 0 }));
00742                           }
00743                           else
00744                           {
00745                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, -1.5
      }));
00746 
00747                               barData.Add(new double[] { underflowCount, -1.5 });
00748                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, -0.5 }));
00749                           }
00750                       }
00751 
00752                       if (overflowCount > 0)
00753                       {
00754                           if (vertical)
00755                           {
00756                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { binCount +
      0.5, 0 }));
00757 
00758                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { binCount -
      0.5, 0 }));
00759                               barData.Add(new double[] { binCount + 0.5, overflowCount });
00760                           }
00761                           else
00762                           {
00763                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0,
      binCount + 0.5 }));
00764 
00765                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, binCount -
      0.5 }));
00766                               barData.Add(new double[] { overflowCount, binCount + 0.5 });
00767                           }
00768                       }
00769 
00770                       Bars overUnderBars = new Bars(barData,
      plot.GetFirst<ICoordinateSystem<IReadOnlyList<double>>(), vertical) { PresentationAttributes =
      dataPresentationAttributes, GetBaseline = plot.GetFirst<Bars>().GetBaseline };
00771 
00772                       plot.AddPlotElement(overUnderBars);
00773 
00774                       Func<int, IReadOnlyList<double>, object> prevLabel = xLabels.Label;
00775                       Func<int, IReadOnlyList<double>, Point> prevMargin = xLabels.Margin;
00776 
00777                       if (vertical)
00778                       {
00779                           xLabels.Label = (i, x) =>
00780                           {
00781                               if (x[0] == -1.5)
00782                               {
00783                                   return "(-∞, " +
      underflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + "]";
00784                               }
00785                               else if (x[0] == binCount + 0.5)
00786                               {
00787                                   return "[" +
```

```
    overflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", +∞)";
00788                           }
00789                           else
00790                           {
00791                               return prevLabel(i, x);
00792                           }
00793                       };
00794
00795                       xLabels.Margin = (i, x) =>
00796                       {
00797                           if (x[0] == -1.5 || x[0] == binCount + 0.5)
00798                           {
00799                               return new Point(0, 20);
00800                           }
00801                           else
00802                           {
00803                               return prevMargin(i, x);
00804                           }
00805                       };
00806                   }
00807                   else
00808                   {
00809                       xLabels.Label = (i, x) =>
00810                       {
00811                           if (x[1] == -1.5)
00812                           {
00813                               return "(-∞, " +
    underflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + "]";
00814                           }
00815                           else if (x[1] == binCount + 0.5)
00816                           {
00817                               return "[" +
    overflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", +∞)";
00818                           }
00819                           else
00820                           {
00821                               return prevLabel(i, x);
00822                           }
00823                       };
00824
00825                       xLabels.Margin = (i, x) =>
00826                       {
00827                           if (x[1] == -1.5 || x[1] == binCount + 0.5)
00828                           {
00829                               return new Point(-20, 0);
00830                           }
00831                           else
00832                           {
00833                               return prevMargin(i, x);
00834                           }
00835                       };
00836
00837                   }
00838
00839                   xLabels.Data = xLabelData;
00840
00841                   double ground = 0;
00842
00843                   if (vertical)
00844                   {
00845                       ground = plot.GetFirst<ContinuousAxisTicks>().StartPoint[1];
00846                   }
00847                   else
00848                   {
00849                       ground = plot.GetAll<ContinuousAxisTicks>().ElementAt(1).StartPoint[0];
00850                   }
00851
00852                   double[] tickStart = null;
00853                   double[] tickEnd = null;
00854
00855                   if (overflowCount > 0 && underflowCount > 0)
00856                   {
00857                       if (vertical)
00858                       {
00859                           tickStart = new double[] { -1.5, ground };
00860                           tickEnd = new double[] { binCount + 0.5, ground };
00861                       }
00862                       else
00863                       {
00864                           tickStart = new double[] { ground, -1.5 };
00865                           tickEnd = new double[] { ground, binCount + 0.5 };
00866                       }
00867                   }
00868                   else if (overflowCount > 0)
00869                   {
00870                       if (vertical)
00871                       {
```

```
00872                           tickStart = new double[] { binCount + 0.5, ground };
00873                           tickEnd = new double[] { binCount + 0.50000001, ground };
00874                       }
00875                       else
00876                       {
00877                           tickStart = new double[] { ground, binCount + 0.5 };
00878                           tickEnd = new double[] { ground, binCount + 0.50000001 };
00879                       }
00880                   }
00881                   else if (underflowCount > 0)
00882                   {
00883                       if (vertical)
00884                       {
00885                           tickStart = new double[] { -1.5, ground };
00886                           tickEnd = new double[] { -1.50000001, ground };
00887                       }
00888                       else
00889                       {
00890                           tickStart = new double[] { ground, -1.5 };
00891                           tickEnd = new double[] { ground, -1.50000001 };
00892                       }
00893                   }
00894
00895                   ContinuousAxisTicks overUnderTicks = new ContinuousAxisTicks(tickStart, tickEnd,
      plot.GetFirst<IContinuousCoordinateSystem>()) { IntervalCount = 1, SizeAbove = i => 3, SizeBelow = i
      => 3 };
00896
00897                   plot.AddPlotElement(overUnderTicks);
00898               }
00899
00900               return plot;
00901           }
00902
00903           private static Plot StackedHistogram(IReadOnlyList<IReadOnlyList<double>> data, bool
      vertical = true, double margin = 0.25, double width = 350, double height = 250,
00904           PlotElementPresentationAttributes axisPresentationAttributes = null,
00905           double axisArrowSize = 3,
00906           PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00907           PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00908           string xAxisTitle = null,
00909           string yAxisTitle = null,
00910           string title = null,
00911           PlotElementPresentationAttributes titlePresentationAttributes = null,
00912           PlotElementPresentationAttributes gridPresentationAttributes = null,
00913           IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00914           IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00915           {
00916               if (axisPresentationAttributes == null)
00917               {
00918                   axisPresentationAttributes = new PlotElementPresentationAttributes();
00919               }
00920
00921               if (gridPresentationAttributes == null)
00922               {
00923                   gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00924               }
00925
00926               if (axisLabelPresentationAttributes == null)
00927               {
00928                   axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00929               }
00930
00931               if (axisTitlePresentationAttributes == null)
00932               {
00933                   axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00934               }
00935
00936               if (dataPresentationAttributes == null)
00937               {
00938                   dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00939                                                                                          new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00940                                                                                          new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00941                                                                                          new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00942                                                                                          new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
```

```
      180, 233)) },
00943                                                                                 new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00944                                                                                 new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00945                 }
00946
00947                 double baselineValue = 0;
00948
00949                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00950                 {
00951                     baselineValue = 1;
00952                 }
00953
00954                 StackedBars bars;
00955
00956                 if (baselineValue == 0)
00957                 {
00958                     bars = new StackedBars(data, coordinateSystem, vertical) { PresentationAttributes
      = dataPresentationAttributes, Margin = margin };
00959                 }
00960                 else
00961                 {
00962                     bars = new StackedBars(data, new Comparison<IReadOnlyList<double»(vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x, y) =>
      Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double»(pt => new
      double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] }, coordinateSystem) {
      PresentationAttributes = dataPresentationAttributes, Margin = margin, Vertical = vertical };
00963                 }
00964
00965                 double[][] actualData = new double[data.Count][];
00966
00967                 for (int i = 0; i < data.Count; i++)
00968                 {
00969                     if (vertical)
00970                     {
00971                         double sum = 0;
00972
00973                         for (int j = 1; j < data[i].Count; j++)
00974                         {
00975                             sum += data[i][j];
00976                         }
00977
00978                         actualData[i] = new double[] { data[i][0], sum };
00979                     }
00980                     else
00981                     {
00982                         double sum = data[i][0];
00983
00984                         for (int j = 2; j < data[i].Count; j++)
00985                         {
00986                             sum += data[i][j];
00987                         }
00988
00989                         actualData[i] = new double[] { sum, data[i][1] };
00990                     }
00991                 }
00992
00993
00994                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(actualData);
00995
00996                 double dataMinX = minX;
00997                 double dataMinY = minY;
00998                 double dataMaxX = maxX;
00999                 double dataMaxY = maxY;
01000
01001                 if (coordinateSystem == null)
01002                 {
01003                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
01004
01005                     if (vertical)
01006                     {
01007                         minY = Math.Min(minY, 0);
01008                         maxY = Math.Max(maxY, 0);
01009                         rangeY = maxY - minY;
01010
01011                         linCoords.MinY = minY - rangeY * 0.1;
01012                         linCoords.MaxY = maxY + rangeY * 0.1;
01013
01014
01015                         if (bars.Data.Count >= 2)
```

```
01016                           {
01017                               IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01018                               IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01019 
01020                               IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01021                               IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01022 
01023                               minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
01024                               maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
01025 
01026                               rangeX = maxX - minX;
01027                               linCoords.MinX = minX;
01028                               linCoords.MaxX = maxX;
01029                           }
01030                       }
01031                       else
01032                       {
01033                           minX = Math.Min(minX, 0);
01034                           maxX = Math.Max(maxX, 0);
01035                           rangeX = maxX - minX;
01036 
01037                           linCoords.MinX = minX - rangeX * 0.1;
01038                           linCoords.MaxX = maxX + rangeX * 0.1;
01039 
01040                           if (bars.Data.Count >= 2)
01041                           {
01042                               IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01043                               IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01044 
01045                               IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01046                               IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01047 
01048                               minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
01049                               maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
01050 
01051                               rangeY = maxY - minY;
01052                               linCoords.MinY = minY;
01053                               linCoords.MaxY = maxY;
01054                           }
01055                       }
01056 
01057                       coordinateSystem = linCoords;
01058                       bars.CoordinateSystem = coordinateSystem;
01059                   }
01060                   else
01061                   {
01062                       if (vertical)
01063                       {
01064                           minY = Math.Min(minY, baselineValue);
01065                           rangeY = maxY - minY;
01066 
01067                           if (bars.Data.Count >= 2)
01068                           {
01069                               IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01070                               IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01071 
01072                               IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01073                               IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01074 
01075                               minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
01076                               maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
01077 
01078                               rangeX = maxX - minX;
01079                           }
01080                       }
01081                       else
01082                       {
01083                           minX = Math.Min(minX, baselineValue);
01084                           rangeX = maxX - minX;
01085 
01086                           if (bars.Data.Count >= 2)
01087                           {
01088                               IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01089                               IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01090 
01091                               IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01092                               IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01093 
01094                               minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
01095                               maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
01096 
01097                               rangeY = maxY - minY;
01098                           }
01099                       }
01100                   }
01101 
01102 
```

```
01103
01104                 if (titlePresentationAttributes == null)
01105                 {
01106                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
01107                 }
01108
01109                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
01110                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
01111                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
01112                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
01113
01114                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01115                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01116                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01117                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01118
01119                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01120                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01121                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01122                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01123
01124
01125                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
01126                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
01127                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
01128                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
01129
01130                 if (vertical)
01131                 {
01132                     Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
01133                     Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
01134
01135                     Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
01136                     Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
01137
01138                     Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
01139                     Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
01140
01141                     Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
01142                     Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
01143
01144                     p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
01145                     p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
01146                     p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
01147                     p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
01148                 }
01149                 else
01150                 {
01151                     Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
```

```
01152                     Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
01153
01154                     Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
01155                     Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
01156
01157                     Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
01158                     Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
01159
01160                     Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
01161                     Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
      });
01162
01163                     p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
01164                     p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
01165                     p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
01166                     p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
01167                 }
01168
01169                 Grid grid;
01170
01171                 if (vertical)
01172                 {
01173                     grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
01174                 }
01175                 else
01176                 {
01177                     grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
01178                 }
01179
01180                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
01181                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
01182
01183                 int every = (int)Math.Ceiling((double)data.Count / 5);
01184                 int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
01185
01186                 ContinuousAxisTicks xTicks;
01187                 ContinuousAxisTicks yTicks;
01188
01189                 if (vertical)
01190                 {
01191                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
01192                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
01193                 }
01194                 else
01195                 {
01196                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
01197                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
01198                 }
01199
01200                 ContinuousAxisLabels xLabels;
01201                 ContinuousAxisLabels yLabels;
01202
01203                 if (vertical)
01204                 {
01205                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
01206                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
01207
01208                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      xLabels.TextFormat;
01209                     xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
01210                 }
01211                 else
01212                 {
01213                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
```

```
    TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
01214                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
    TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
01215
01216                   Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
    yLabels.TextFormat;
01217                   yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
    :  null;
01218               }
01219
01220               Graphics xLabelsSize = new Graphics();
01221               xLabels.Plot(xLabelsSize);
01222               double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
01223
01224               Graphics yLabelsSize = new Graphics();
01225               yLabels.Plot(yLabelsSize);
01226               double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
01227
01228               ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
    marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
    Alignment = TextAnchors.Center };
01229               ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
    marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
    Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
01230
01231               TextLabel<IReadOnlyList<double» titleLabel = new
    TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
    topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
    TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
01232
01233               Plot tbr = new Plot();
01234               tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
    yTitle, bars, titleLabel);
01235
01236               return tbr;
01237           }
01238
01239 /// <summary>
01240 /// Create a new distribution plot.
01241 /// </summary>
01242 /// <param name="data">The data whose distribution will be plotted.</param>
01243 /// <param name="vertical">If this is <see langword="true"/> (the default), the distribution goes from
    the X axis up to the sampled values.  If this is <see langword="false"/>, the distribution goes from
    the Y axis to the sampled values.</param>
01244 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
    determined automatically using the Freedman-Diaconis rule.</param>
01245 /// <param name="smooth">If this is <see langword="false"/> (the default), the values are joined by a
    polyline.  If this is <see langword="true"/>, the values are joined by a smooth spline passing through
    all of them.</param>
01246 /// <param name="width">The width of the plot.</param>
01247 /// <param name="height">The height of the plot.</param>
01248 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
01249 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
01250 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
01251 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
01252 /// <param name="xAxisTitle">Title for the X axis.</param>
01253 /// <param name="yAxisTitle">Title for the Y axis.</param>
01254 /// <param name="title">Title for the plot.</param>
01255 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
01256 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
01257 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
01258 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
01259 /// <returns>A <see cref="Plot"/> containing the distribution plot.</returns>
01260           public static Plot Distribution(IReadOnlyList<double> data, bool vertical = true, int
    binCount = -1, bool smooth = false, double width = 350, double height = 250,
01261               PlotElementPresentationAttributes axisPresentationAttributes = null,
01262               double axisArrowSize = 3,
01263               PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
01264               PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
01265               string xAxisTitle = null,
01266               string yAxisTitle = null,
01267               string title = null,
01268               PlotElementPresentationAttributes titlePresentationAttributes = null,
01269               PlotElementPresentationAttributes gridPresentationAttributes = null,
01270               PlotElementPresentationAttributes dataPresentationAttributes = null,
01271               IContinuousInvertibleCoordinateSystem coordinateSystem = null)
01272           {
01273               if (dataPresentationAttributes == null)
01274               {
01275                   dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
    new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 2, Fill = new
    SolidColourBrush(Colour.FromRgb(0, 114, 178).WithAlpha(0.5)) };
01276               }
01277
01278               double min = double.MaxValue;
```

```
01279                 double max = double.MinValue;
01280
01281                 for (int i = 0; i < data.Count; i++)
01282                 {
01283                     min = Math.Min(min, data[i]);
01284                     max = Math.Max(max, data[i]);
01285                 }
01286
01287                 if (binCount <= 1)
01288                 {
01289                     (double _, double _, double iqr) = IQR(data);
01290                     double h2 = 2 * iqr / Math.Pow(data.Count, 1.0 / 3.0);
01291
01292                     if (h2 > 0)
01293                     {
01294                         binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
01295                     }
01296                     else
01297                     {
01298                         binCount = 2;
01299                     }
01300                 }
01301
01302                 int[] bins = new int[binCount];
01303
01304                 if (max > min)
01305                 {
01306                     for (int i = 0; i < data.Count; i++)
01307                     {
01308                         int index = (int)Math.Min(binCount - 1, Math.Floor((data[i] - min) / (max -
      min) * binCount));
01309
01310                         bins[index]++;
01311                     }
01312                 }
01313                 else
01314                 {
01315                     bins[0] = data.Count;
01316                 }
01317
01318                 double[][] binnedData = new double[bins.Length][];
01319
01320                 for (int i = 0; i < bins.Length; i++)
01321                 {
01322                     double binStart = min + (max - min) / binCount * i;
01323                     double binEnd = min + (max - min) / binCount * (i + 1);
01324
01325                     if (vertical)
01326                     {
01327                         binnedData[i] = new double[] { (binStart + binEnd) * 0.5, bins[i] };
01328                     }
01329                     else
01330                     {
01331                         binnedData[i] = new double[] { bins[i], (binStart + binEnd) * 0.5 };
01332                     }
01333                 }
01334
01335                 Plot plot = Create.AreaChart(binnedData, vertical, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
01336
01337                 return plot;
01338             }
01339
01340 /// <summary>
01341 /// Create a new stacked distribution plot.
01342 /// </summary>
01343 /// <param name="data">The data whose distribution will be plotted.</param>
01344 /// <param name="vertical">If this is <see langword="true"/> (the default), the distributions go from
      the X axis up to the sampled values.  If this is <see langword="false"/>, the distributions go from
      the Y axis to the sampled values.</param>
01345 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
      determined automatically using the Freedman-Diaconis rule.</param>
01346 /// <param name="smooth">If this is <see langword="false"/> (the default), the values are joined by a
      polyline.  If this is <see langword="true"/>, the values are joined by a smooth spline passing through
      all of them.</param>
01347 /// <param name="normalisationMode">The kind of normalisation to use to make the distributions
      comparable.</param>
01348 /// <param name="width">The width of the plot.</param>
01349 /// <param name="height">The height of the plot.</param>
01350 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
01351 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
01352 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
01353 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
01354 /// <param name="xAxisTitle">Title for the X axis.</param>
01355 /// <param name="yAxisTitle">Title for the Y axis.</param>
```

```
01356 /// <param name="title">Title for the plot.</param>
01357 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
01358 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
01359 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
01360 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01361 /// <returns>A <see cref="Plot"/> containing the stacked distribution plot.</returns>
01362             public static Plot StackedDistribution(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, int binCount = -1, bool smooth = false, NormalisationMode normalisationMode =
      NormalisationMode.Area, double width = 350, double height = 250,
01363                 PlotElementPresentationAttributes axisPresentationAttributes = null,
01364                 double axisArrowSize = 3,
01365                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
01366                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
01367                 string xAxisTitle = null,
01368                 string yAxisTitle = null,
01369                 string title = null,
01370                 PlotElementPresentationAttributes titlePresentationAttributes = null,
01371                 PlotElementPresentationAttributes gridPresentationAttributes = null,
01372                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
01373                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
01374             {
01375                 if (dataPresentationAttributes == null)
01376                 {
01377                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 114,
      178).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
01378                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(213, 94,
      0).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
01379                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(204, 121,
      167).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
01380                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(230, 159,
      0).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
01381                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(86, 180,
      233).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
01382                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 158,
      115).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
01383                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(240, 228,
      66).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
01384                 }
01385
01386                 bool wasCoordinateSystemNull = coordinateSystem == null;
01387
01388                 double[][][] allBinnedData = new double[data.Count][][];
01389
01390                 double overallMinX = double.MaxValue;
01391                 double overallMaxX = double.MinValue;
01392
01393                 double overallMaxY = double.MinValue;
01394
01395                 double minX0 = double.MaxValue;
01396                 double maxX0 = double.MinValue;
01397                 double maxY0 = double.MaxValue;
01398
01399                 for (int j = 0; j < data.Count; j++)
01400                 {
01401
01402                     double min = double.MaxValue;
01403                     double max = double.MinValue;
01404
01405                     for (int i = 0; i < data[j].Count; i++)
01406                     {
01407                         min = Math.Min(min, data[j][i]);
01408                         max = Math.Max(max, data[j][i]);
01409                     }
01410
01411                     if (binCount <= 1)
01412                     {
01413                         (double _, double _, double iqr) = IQR(data[j]);
01414                         double h2 = 2 * iqr / Math.Pow(data[j].Count, 1.0 / 3.0);
01415
01416                         if (h2 > 0)
01417                         {
01418                             binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
01419                         }
01420                         else
01421                         {
01422                             binCount = 2;
01423                         }
01424                     }
01425
```

```
01426                     int[] bins = new int[binCount];
01427
01428                     if (max > min)
01429                     {
01430                         for (int i = 0; i < data[j].Count; i++)
01431                         {
01432                             int index = (int)Math.Min(binCount - 1, Math.Floor((data[j][i] - min) /
      (max - min) * binCount));
01433
01434                             bins[index]++;
01435                         }
01436                     }
01437                     else
01438                     {
01439                         bins[0] = data[j].Count;
01440                     }
01441
01442                     double[][] binnedData = new double[bins.Length][];
01443
01444                     double maxBin = 1;
01445
01446                     if (normalisationMode == NormalisationMode.Maximum)
01447                     {
01448                         maxBin = bins.Max();
01449                     }
01450                     else if (normalisationMode == NormalisationMode.None)
01451                     {
01452                         maxBin = 1;
01453                     }
01454                     else if (normalisationMode == NormalisationMode.Area)
01455                     {
01456                         maxBin = (max - min) / binCount * data[j].Count;
01457                     }
01458
01459                     for (int i = 0; i < bins.Length; i++)
01460                     {
01461                         double binStart = min + (max - min) / binCount * i;
01462                         double binEnd = min + (max - min) / binCount * (i + 1);
01463
01464                         if (vertical)
01465                         {
01466                             binnedData[i] = new double[] { (binStart + binEnd) * 0.5, bins[i] / maxBin
      };
01467                         }
01468                         else
01469                         {
01470                             binnedData[i] = new double[] { bins[i] / maxBin, (binStart + binEnd) * 0.5
      };
01471                         }
01472
01473                         overallMaxY = Math.Max(overallMaxY, bins[i] / maxBin);
01474                         overallMaxX = Math.Max(overallMaxX, (binStart + binEnd) * 0.5);
01475                         overallMinX = Math.Min(overallMinX, (binStart + binEnd) * 0.5);
01476                     }
01477
01478                     if (j == 0)
01479                     {
01480                         maxX0 = overallMaxX;
01481                         minX0 = overallMinX;
01482                         maxY0 = overallMaxY;
01483                     }
01484
01485                     allBinnedData[j] = binnedData;
01486                 }
01487
01488                 List<double[]> joinedBinnedData = new List<double[]>((from el in allBinnedData select
      el.Length).Sum());
01489
01490                 for (int i = 0; i < allBinnedData.Length; i++)
01491                 {
01492                     joinedBinnedData.AddRange(allBinnedData[i]);
01493                 }
01494
01495                 Plot plot = Create.AreaChart(joinedBinnedData, vertical, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes[0], coordinateSystem);
01496
01497                 Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline =
      plot.GetFirst<Area<IReadOnlyList<double»>().GetBaseline;
01498
01499                 plot.RemovePlotElement(plot.GetFirst<Area<IReadOnlyList<double»>());
01500
01501                 ICoordinateSystem<IReadOnlyList<double» currCoordinateSystem =
      plot.GetFirst<ICoordinateSystem<IReadOnlyList<double»>();
01502
01503                 for (int i = 0; i < allBinnedData.Length; i++)
```

```
01504                 {
01505                     Area<IReadOnlyList<double» area = new Area<IReadOnlyList<double»(allBinnedData[i],
      getBaseline, currCoordinateSystem) { Smooth = smooth, PresentationAttributes =
      dataPresentationAttributes[i % dataPresentationAttributes.Count] };
01506                     plot.AddPlotElement(area);
01507                 }
01508 
01509                 return plot;
01510             }
01511         }
01512     }
01513 }
```

8.35 Plot.Lines.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new line chart containing multiple lines.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00033 /// <param name="width">The width of the plot.</param>
00034 /// <param name="height">The height of the plot.</param>
00035 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00036 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00037 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00038 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00039 /// <param name="xAxisTitle">Title for the X axis.</param>
00040 /// <param name="yAxisTitle">Title for the Y axis.</param>
00041 /// <param name="title">Title for the plot.</param>
00042 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00043 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00044 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00045 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00046 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00047 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00048 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00049 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00050             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<IReadOnlyList<double»> data,
      bool smooth = false, double width = 350, double height = 250,
00051                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00052                 double axisArrowSize = 3,
00053                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00054                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00055                 string xAxisTitle = null,
00056                 string yAxisTitle = null,
00057                 string title = null,
00058                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00059                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00060                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00061                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00062                 IReadOnlyList<double> pointSizes = null,
00063                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00064                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
```

```
00065             {
00066                 if (axisPresentationAttributes == null)
00067                 {
00068                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00069                 }
00070 
00071                 if (gridPresentationAttributes == null)
00072                 {
00073                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00074                 }
00075 
00076                 if (axisLabelPresentationAttributes == null)
00077                 {
00078                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00079                 }
00080 
00081                 if (axisTitlePresentationAttributes == null)
00082                 {
00083                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00084                 }
00085 
00086                 if (linePresentationAttributes == null)
00087                 {
00088                     linePresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill
      = null },
00089                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill
      = null },
00090                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)),
      Fill = null },
00091                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), Fill
      = null },
00092                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)),
      Fill = null },
00093                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), Fill
      = null },
00094                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)),
      Fill = null } };
00095                 }
00096 
00097                 if (pointPresentationAttributes == null)
00098                 {
00099                     pointPresentationAttributes = (from el in linePresentationAttributes select new
      PlotElementPresentationAttributes() { Stroke = null, Fill = el.Stroke }).ToList();
00100                 }
00101 
00102                 if (dataPointElements == null)
00103                 {
00104                     dataPointElements = new PathDataPointElement[] { new PathDataPointElement() };
00105                 }
00106 
00107                 List<IReadOnlyList<double» allData = new List<IReadOnlyList<double»();
00108 
00109                 for (int i = 0; i < data.Count; i++)
00110                 {
00111                     allData.AddRange(data[i]);
00112                 }
00113 
00114                 if (coordinateSystem == null)
00115                 {
00116                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00117                 }
00118 
00119                 if (titlePresentationAttributes == null)
00120                 {
00121                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00122                 }
00123 
00124                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00125 
00126                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00127                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00128                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00129                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
```

```
00130
00131                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00132                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00133                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00134                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00135
00136                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00137                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00138                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00139                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00140
00141
00142                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00143                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00144                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00145                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00146
00147                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00148                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00149
00150                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00151                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00152
00153                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00154                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00155
00156                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00157                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00158
00159                 Graphics xLabelsSize = new Graphics();
00160                 xLabels.Plot(xLabelsSize);
00161                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00162
00163                 Graphics yLabelsSize = new Graphics();
00164                 yLabels.Plot(yLabelsSize);
00165                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00166
00167                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00168                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00169
00170                 List<IPlotElement> otherElements = new List<IPlotElement>();
00171
00172                 for (int i = 0; i < data.Count; i++)
00173                 {
00174                     DataLine<IReadOnlyList<double» line1 = new DataLine<IReadOnlyList<double»(data[i],
      coordinateSystem) { PresentationAttributes = linePresentationAttributes[i %
      linePresentationAttributes.Count], Smooth = smooth };
00175
00176                     otherElements.Add(line1);
```

```
00177
00178                   if (pointSizes != null && pointSizes.Count > 0 && pointSizes[i % pointSizes.Count]
    > 0)
00179                   {
00180                       ScatterPoints<IReadOnlyList<double» points1 = new
    ScatterPoints<IReadOnlyList<double»(data[i], coordinateSystem) { PresentationAttributes =
    pointPresentationAttributes[i % pointPresentationAttributes.Count], DataPointElement =
    dataPointElements[i % dataPointElements.Count], Size = pointSizes[i % pointSizes.Count] };
00181                       otherElements.Add(points1);
00182                   }
00183
00184               }
00185
00186               TextLabel<IReadOnlyList<double» titleLabel = new
    TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
    topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
    TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00187
00188               Plot tbr = new Plot();
00189               tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
    xTitle, yTitle);
00190
00191               tbr.AddPlotElements(otherElements);
00192
00193               tbr.AddPlotElement(titleLabel);
00194
00195               return tbr;
00196           }
00197
00198 /// <summary>
00199 /// Create a new line chart.
00200 /// </summary>
00201 /// <param name="data">The data to plot.</param>
00202 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
    joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
    through them.</param>
00203 /// <param name="width">The width of the plot.</param>
00204 /// <param name="height">The height of the plot.</param>
00205 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00206 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00207 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00208 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00209 /// <param name="xAxisTitle">Title for the X axis.</param>
00210 /// <param name="yAxisTitle">Title for the Y axis.</param>
00211 /// <param name="title">Title for the plot.</param>
00212 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00213 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00214 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00215 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00216 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00217 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00218 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00219 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00220           public static Plot LineChart(IReadOnlyList<IReadOnlyList<double» data, bool smooth =
    false, double width = 350, double height = 250,
00221               PlotElementPresentationAttributes axisPresentationAttributes = null,
00222               double axisArrowSize = 3,
00223               PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00224               PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00225               string xAxisTitle = null,
00226               string yAxisTitle = null,
00227               string title = null,
00228               PlotElementPresentationAttributes titlePresentationAttributes = null,
00229               PlotElementPresentationAttributes gridPresentationAttributes = null,
00230               PlotElementPresentationAttributes linePresentationAttributes = null,
00231               PlotElementPresentationAttributes pointPresentationAttributes = null,
00232               double pointSize = 0,
00233               IDataPointElement dataPointElement = null,
00234               IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00235           {
00236               return LineCharts(new IReadOnlyList<IReadOnlyList<double»[] { data }, smooth, width,
    height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
    axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
    gridPresentationAttributes, linePresentationAttributes == null ?  null :  new
    PlotElementPresentationAttributes[] { linePresentationAttributes }, pointPresentationAttributes ==
    null ?  null :  new PlotElementPresentationAttributes[] { pointPresentationAttributes }, new double[]
    { pointSize }, dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement },
    coordinateSystem);
00237           }
00238
00239 /// <summary>
00240 /// Create a new line chart.
00241 /// </summary>
00242 /// <param name="data">The data to plot.</param>
00243 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
    joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
```

```
through them.</param>
00244 /// <param name="width">The width of the plot.</param>
00245 /// <param name="height">The height of the plot.</param>
00246 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00247 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00248 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00249 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00250 /// <param name="xAxisTitle">Title for the X axis.</param>
00251 /// <param name="yAxisTitle">Title for the Y axis.</param>
00252 /// <param name="title">Title for the plot.</param>
00253 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00254 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00255 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00256 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00257 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00258 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00259 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00260 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00261             public static Plot LineChart(IReadOnlyList<(double, double)> data, bool smooth = false,
double width = 350, double height = 250,
00262                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00263                 double axisArrowSize = 3,
00264                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00265                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00266                 string xAxisTitle = null,
00267                 string yAxisTitle = null,
00268                 string title = null,
00269                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00270                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00271                 PlotElementPresentationAttributes linePresentationAttributes = null,
00272                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00273                 double pointSize = 0,
00274                 IDataPointElement dataPointElement = null,
00275                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00276             {
00277                 return LineCharts(new IReadOnlyList<IReadOnlyList<double>>[] { (from el in data select
new double[] { el.Item1, el.Item2 }).ToArray() }, smooth, width, height, axisPresentationAttributes,
axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes, linePresentationAttributes
== null ?  null :  new PlotElementPresentationAttributes[] { linePresentationAttributes },
pointPresentationAttributes == null ?  null :  new PlotElementPresentationAttributes[] {
pointPresentationAttributes }, new double[] { pointSize }, dataPointElement == null ?  null :  new
IDataPointElement[] { dataPointElement }, coordinateSystem);
00278             }
00279
00280 /// <summary>
00281 /// Create a new line chart.
00282 /// </summary>
00283 /// <param name="data">The data to plot.</param>
00284 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
through them.</param>
00285 /// <param name="width">The width of the plot.</param>
00286 /// <param name="height">The height of the plot.</param>
00287 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00288 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00289 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00290 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00291 /// <param name="xAxisTitle">Title for the X axis.</param>
00292 /// <param name="yAxisTitle">Title for the Y axis.</param>
00293 /// <param name="title">Title for the plot.</param>
00294 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00295 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00296 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00297 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00298 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00299 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00300 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00301 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00302             public static Plot LineChart(IReadOnlyList<double> data, bool smooth = false, double width
= 350, double height = 250,
00303                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00304                 double axisArrowSize = 3,
00305                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00306                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00307                 string xAxisTitle = null,
00308                 string yAxisTitle = null,
00309                 string title = null,
00310                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00311                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00312                 PlotElementPresentationAttributes linePresentationAttributes = null,
00313                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00314                 double pointSize = 0,
00315                 IDataPointElement dataPointElement = null,
00316                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
```

```
00317             {
00318                 double[][] actualData = new double[data.Count][];
00319 
00320                 for (int i = 0; i < data.Count; i++)
00321                 {
00322                     actualData[i] = new double[] { i + 1, data[i] };
00323                 }
00324 
00325                 Plot plot = LineChart(actualData, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSize, dataPointElement,
      coordinateSystem);
00326 
00327                 Plot tbr = new Plot();
00328 
00329                 for (int i = 0; i < plot.PlotElements.Count; i++)
00330                 {
00331                     if (i != 2 && i != 4 && i != 6)
00332                     {
00333                         tbr.AddPlotElement(plot.PlotElements[i]);
00334                     }
00335                 }
00336 
00337                 return tbr;
00338             }
00339 
00340 /// <summary>
00341 /// Create a new line chart containing two lines.
00342 /// </summary>
00343 /// <param name="data1">The data to plot for the first line.</param>
00344 /// <param name="data2">The data to plot for the second line.</param>
00345 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00346 /// <param name="width">The width of the plot.</param>
00347 /// <param name="height">The height of the plot.</param>
00348 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00349 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00350 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00351 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00352 /// <param name="xAxisTitle">Title for the X axis.</param>
00353 /// <param name="yAxisTitle">Title for the Y axis.</param>
00354 /// <param name="title">Title for the plot.</param>
00355 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00356 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00357 /// <param name="line1PresentationAttributes">Presentation attributes for the first line.</param>
00358 /// <param name="point1PresentationAttributes">Presentation attributes for the sampled points for the
      first line.</param>
00359 /// <param name="point1Size">Size of the symbols drawn at the sampled points for the first
      line.</param>
00360 /// <param name="line2PresentationAttributes">Presentation attributes for the second line.</param>
00361 /// <param name="point2PresentationAttributes">Presentation attributes for the sampled points for the
      second line.</param>
00362 /// <param name="point2Size">Size of the symbols drawn at the sampled points for the second
      line.</param>
00363 /// <param name="dataPointElement1">Symbol drawn at the sampled points for the first line.</param>
00364 /// <param name="dataPointElement2">Symbol drawn at the sampled points for the second line.</param>
00365 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00366 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00367             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<double» data1,
      IReadOnlyList<IReadOnlyList<double» data2, bool smooth = false, double width = 350, double height =
      250,
00368                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00369                 double axisArrowSize = 3,
00370                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00371                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00372                 string xAxisTitle = null,
00373                 string yAxisTitle = null,
00374                 string title = null,
00375                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00376                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00377                 PlotElementPresentationAttributes line1PresentationAttributes = null,
00378                 PlotElementPresentationAttributes point1PresentationAttributes = null,
00379                 double point1Size = 0,
00380                 PlotElementPresentationAttributes line2PresentationAttributes = null,
00381                 PlotElementPresentationAttributes point2PresentationAttributes = null,
00382                 double point2Size = 0,
00383                 IDataPointElement dataPointElement1 = null,
00384                 IDataPointElement dataPointElement2 = null,
00385                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00386             {
00387                 if (line1PresentationAttributes == null)
00388                 {
00389                     line1PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill = null };
```

```
00390                 }
00391
00392                 if (point1PresentationAttributes == null)
00393                 {
00394                     point1PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = line1PresentationAttributes.Stroke };
00395                 }
00396
00397                 if (line2PresentationAttributes == null)
00398                 {
00399                     line2PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill = null };
00400                 }
00401
00402                 if (point2PresentationAttributes == null)
00403                 {
00404                     point2PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = line2PresentationAttributes.Stroke };
00405                 }
00406
00407                 if (dataPointElement1 == null)
00408                 {
00409                     dataPointElement1 = new PathDataPointElement();
00410                 }
00411
00412                 if (dataPointElement2 == null)
00413                 {
00414                     dataPointElement2 = new PathDataPointElement();
00415                 }
00416
00417                 return LineCharts(new IReadOnlyList<IReadOnlyList<double>[] { data1, data2 }, smooth,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, new PlotElementPresentationAttributes[] { line1PresentationAttributes,
      line2PresentationAttributes }, new PlotElementPresentationAttributes[] { point1PresentationAttributes,
      point2PresentationAttributes }, new double[] { point1Size, point2Size }, new IDataPointElement[] {
      dataPointElement1, dataPointElement2 }, coordinateSystem);
00418             }
00419
00420 /// <summary>
00421 /// Create a new line chart containing two lines.
00422 /// </summary>
00423 /// <param name="data1">The data to plot for the first line.</param>
00424 /// <param name="data2">The data to plot for the second line.</param>
00425 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00426 /// <param name="width">The width of the plot.</param>
00427 /// <param name="height">The height of the plot.</param>
00428 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00429 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00430 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00431 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00432 /// <param name="xAxisTitle">Title for the X axis.</param>
00433 /// <param name="yAxisTitle">Title for the Y axis.</param>
00434 /// <param name="title">Title for the plot.</param>
00435 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00436 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00437 /// <param name="line1PresentationAttributes">Presentation attributes for the first line.</param>
00438 /// <param name="point1PresentationAttributes">Presentation attributes for the sampled points for the
      first line.</param>
00439 /// <param name="point1Size">Size of the symbols drawn at the sampled points for the first
      line.</param>
00440 /// <param name="line2PresentationAttributes">Presentation attributes for the second line.</param>
00441 /// <param name="point2PresentationAttributes">Presentation attributes for the sampled points for the
      second line.</param>
00442 /// <param name="point2Size">Size of the symbols drawn at the sampled points for the second
      line.</param>
00443 /// <param name="dataPointElement1">Symbol drawn at the sampled points for the first line.</param>
00444 /// <param name="dataPointElement2">Symbol drawn at the sampled points for the second line.</param>
00445 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00446 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00447             public static Plot LineCharts(IReadOnlyList<double> data1, IReadOnlyList<double> data2,
      bool smooth = false, double width = 350, double height = 250,
00448                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00449                 double axisArrowSize = 3,
00450                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00451                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00452                 string xAxisTitle = null,
00453                 string yAxisTitle = null,
00454                 string title = null,
00455                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00456                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00457                 PlotElementPresentationAttributes line1PresentationAttributes = null,
00458                 PlotElementPresentationAttributes point1PresentationAttributes = null,
00459                 double point1Size = 0,
```

```
00460                 PlotElementPresentationAttributes line2PresentationAttributes = null,
00461                 PlotElementPresentationAttributes point2PresentationAttributes = null,
00462                 double point2Size = 0,
00463                 IDataPointElement dataPointElement1 = null,
00464                 IDataPointElement dataPointElement2 = null,
00465                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00466             {
00467                 double[][] actualData1 = new double[data1.Count][];
00468 
00469                 for (int i = 0; i < data1.Count; i++)
00470                 {
00471                     actualData1[i] = new double[] { i + 1, data1[i] };
00472                 }
00473 
00474                 double[][] actualData2 = new double[data2.Count][];
00475 
00476                 for (int i = 0; i < data2.Count; i++)
00477                 {
00478                     actualData2[i] = new double[] { i + 1, data2[i] };
00479                 }
00480 
00481                 Plot plot = LineCharts(actualData1, actualData2, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, line1PresentationAttributes, point1PresentationAttributes, point1Size,
      line2PresentationAttributes, point2PresentationAttributes, point2Size, dataPointElement1,
      dataPointElement2, coordinateSystem);
00482 
00483                 Plot tbr = new Plot();
00484 
00485                 for (int i = 0; i < plot.PlotElements.Count; i++)
00486                 {
00487                     if (i != 2 && i != 4 && i != 6)
00488                     {
00489                         tbr.AddPlotElement(plot.PlotElements[i]);
00490                     }
00491                 }
00492 
00493                 return tbr;
00494             }
00495 
00496 /// <summary>
00497 /// Create a new line chart containing multiple lines.
00498 /// </summary>
00499 /// <param name="data">The data to plot.</param>
00500 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00501 /// <param name="width">The width of the plot.</param>
00502 /// <param name="height">The height of the plot.</param>
00503 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00504 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00505 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00506 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00507 /// <param name="xAxisTitle">Title for the X axis.</param>
00508 /// <param name="yAxisTitle">Title for the Y axis.</param>
00509 /// <param name="title">Title for the plot.</param>
00510 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00511 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00512 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00513 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00514 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00515 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00516 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00517 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00518             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<(double, double)» data, bool
      smooth = false, double width = 350, double height = 250,
00519                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00520                 double axisArrowSize = 3,
00521                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00522                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00523                 string xAxisTitle = null,
00524                 string yAxisTitle = null,
00525                 string title = null,
00526                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00527                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00528                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00529                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00530                 IReadOnlyList<double> pointSizes = null,
00531                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00532                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00533             {
00534                 return LineCharts((from el in data select (from el2 in el select new double[] {
      el2.Item1, el2.Item2 }).ToArray()).ToArray(), smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
```

```
      linePresentationAttributes, pointPresentationAttributes, pointSizes, dataPointElements,
      coordinateSystem);
00535             }
00536
00537 /// <summary>
00538 /// Create a new line chart containing multiple lines.
00539 /// </summary>
00540 /// <param name="data">The data to plot.</param>
00541 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00542 /// <param name="width">The width of the plot.</param>
00543 /// <param name="height">The height of the plot.</param>
00544 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00545 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00546 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00547 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00548 /// <param name="xAxisTitle">Title for the X axis.</param>
00549 /// <param name="yAxisTitle">Title for the Y axis.</param>
00550 /// <param name="title">Title for the plot.</param>
00551 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00552 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00553 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00554 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00555 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00556 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00557 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00558 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00559             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<double» data, bool smooth =
      false, double width = 350, double height = 250,
00560                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00561                 double axisArrowSize = 3,
00562                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00563                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00564                 string xAxisTitle = null,
00565                 string yAxisTitle = null,
00566                 string title = null,
00567                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00568                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00569                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00570                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00571                 IReadOnlyList<double> pointSizes = null,
00572                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00573                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00574             {
00575                 double[][][] actualData = new double[data.Count][][];
00576
00577                 for (int i = 0; i < data.Count; i++)
00578                 {
00579                     actualData[i] = new double[data[i].Count][];
00580
00581                     for (int j = 0; j < data[i].Count; j++)
00582                     {
00583                         actualData[i][j] = new double[] { j + 1, data[i][j] };
00584                     }
00585                 }
00586
00587
00588                 Plot plot = LineCharts(actualData, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSizes, dataPointElements,
      coordinateSystem);
00589
00590                 Plot tbr = new Plot();
00591
00592                 for (int i = 0; i < plot.PlotElements.Count; i++)
00593                 {
00594                     if (i != 2 && i != 4 && i != 6)
00595                     {
00596                         tbr.AddPlotElement(plot.PlotElements[i]);
00597                     }
00598                 }
00599
00600                 return tbr;
00601             }
00602         }
00603     }
00604 }
```

8.36 Plot.Pie.cs

```
00001 /*
```

```
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new doughnut chart containing multiple doughnuts.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00033 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00034 /// <param name="innerRadius">The radius of the doughnut "hole".</param>
00035 /// <param name="width">The width of the plot.</param>
00036 /// <param name="height">The height of the plot.</param>
00037 /// <param name="title">Title for the plot.</param>
00038 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00039 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00040 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00041 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00042             public static Plot DoughnutCharts(IReadOnlyList<IReadOnlyList<double» data, bool clockwise
      = false, double startAngle = 0, double innerRadius = 0.5, double width = 350, double height = 250,
00043                 string title = null,
00044                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00045                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00046                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00047             {
00048
00049                 if (dataPresentationAttributes == null)
00050                 {
00051                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00052                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00053                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00054                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00055                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
      180, 233)) },
00056                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00057                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00058                 }
00059
00060                 int rowCount = (int)Math.Round(Math.Sqrt(data.Count));
00061
00062                 int[] elementsPerRow = new int[rowCount];
00063
00064                 for (int i = 0; i < rowCount; i++)
00065                 {
00066                     elementsPerRow[i] = data.Count / rowCount;
00067
00068                     if (data.Count - (data.Count / rowCount) * rowCount > i)
00069                     {
00070                         elementsPerRow[i]++;
```

```
00071                       }
00072                   }
00073
00074                   if (coordinateSystem == null)
00075                   {
00076                       double pieWidth = 2.5 * elementsPerRow.Max() + 2;
00077                       double pieHeight = 2.5 * rowCount + 2;
00078
00079                       double aspectRatio = pieWidth / pieHeight;
00080                       double targetAspectRatio = width / height;
00081
00082                       if (targetAspectRatio > aspectRatio)
00083                       {
00084                           pieWidth = targetAspectRatio * pieHeight;
00085                       }
00086                       else
00087                       {
00088                           pieHeight = pieWidth / targetAspectRatio;
00089                       }
00090
00091                       coordinateSystem = new LinearCoordinateSystem2D(-1, pieWidth - 1, -pieHeight + 1,
      1, width, height);
00092                   }
00093
00094                   if (titlePresentationAttributes == null)
00095                   {
00096                       titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00097                   }
00098
00099                   Plot tbr = new Plot();
00100
00101
00102                   for (int i = 0; i < data.Count; i++)
00103                   {
00104                       int column = i;
00105                       int row = 0;
00106
00107                       for (int j = 0; j < rowCount; j++)
00108                       {
00109                           if (elementsPerRow[j] <= column)
00110                           {
00111                               column -= elementsPerRow[j];
00112                               row++;
00113                           }
00114                           else
00115                           {
00116                               break;
00117                           }
00118                       }
00119
00120                       Pie pie = new Pie(data[i], new double[] { column * 2.5 + 1.25 *
      (elementsPerRow.Max() - elementsPerRow[row]), -row * 2.5 }, new double[] { 1, 1 }, coordinateSystem)
00121                       {
00122                           PresentationAttributes = dataPresentationAttributes,
00123                           Clockwise = clockwise,
00124                           StartAngle = startAngle,
00125                           InnerRadius = new double[] { innerRadius, innerRadius }
00126                       };
00127
00128                       tbr.AddPlotElement(pie);
00129                   }
00130
00131                   Point top = coordinateSystem.ToPlotCoordinates(new double[] { (elementsPerRow.Max() -
      1) * 0.5 * 2.5, 1 });
00132                   double[] titleCentre = coordinateSystem.ToDataCoordinates(new Point(top.X, top.Y -
      10));
00133
00134                   TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, titleCentre, coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00135                   tbr.AddPlotElement(titleLabel);
00136
00137
00138                   return tbr;
00139               }
00140
00141 /// <summary>
00142 /// Create a new doughnut chart.
00143 /// </summary>
00144 /// <param name="data">The data to plot.</param>
00145 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00146 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00147 /// <param name="innerRadius">The radius of the doughnut "hole".</param>
```

```
00148 /// <param name="width">The width of the plot.</param>
00149 /// <param name="height">The height of the plot.</param>
00150 /// <param name="title">Title for the plot.</param>
00151 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00152 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00153 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00154 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00155             public static Plot DoughnutChart(IReadOnlyList<double> data, bool clockwise = false,
      double startAngle = 0, double innerRadius = 0.5, double width = 350, double height = 250,
00156              string title = null,
00157              PlotElementPresentationAttributes titlePresentationAttributes = null,
00158              IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00159              IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00160             {
00161                 return DoughnutCharts(new IReadOnlyList<double>[] { data }, clockwise, startAngle,
      innerRadius, width, height, title, titlePresentationAttributes, dataPresentationAttributes,
      coordinateSystem);
00162             }
00163
00164
00165 /// <summary>
00166 /// Create a new pie chart.
00167 /// </summary>
00168 /// <param name="data">The data to plot.</param>
00169 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00170 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00171 /// <param name="width">The width of the plot.</param>
00172 /// <param name="height">The height of the plot.</param>
00173 /// <param name="title">Title for the plot.</param>
00174 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00175 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00176 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00177 /// <returns>A <see cref="Plot"/> containing the pie chart.</returns>
00178             public static Plot PieChart(IReadOnlyList<double> data, bool clockwise = false, double
      startAngle = 0, double width = 350, double height = 250,
00179              string title = null,
00180              PlotElementPresentationAttributes titlePresentationAttributes = null,
00181              IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00182              IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00183             {
00184                 return DoughnutCharts(new IReadOnlyList<double>[] { data }, clockwise, startAngle, 0,
      width, height, title, titlePresentationAttributes, dataPresentationAttributes, coordinateSystem);
00185             }
00186
00187 /// <summary>
00188 /// Create a new pie chart containing multiple pies.
00189 /// </summary>
00190 /// <param name="data">The data to plot.</param>
00191 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00192 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00193 /// <param name="width">The width of the plot.</param>
00194 /// <param name="height">The height of the plot.</param>
00195 /// <param name="title">Title for the plot.</param>
00196 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00197 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00198 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00199 /// <returns>A <see cref="Plot"/> containing the pie chart.</returns>
00200             public static Plot PieCharts(IReadOnlyList<IReadOnlyList<double» data, bool clockwise =
      false, double startAngle = 0, double width = 350, double height = 250,
00201                 string title = null,
00202                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00203                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00204                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00205             {
00206                 return DoughnutCharts(data, clockwise, startAngle, 0, width, height, title,
      titlePresentationAttributes, dataPresentationAttributes, coordinateSystem);
00207             }
00208
00209 /// <summary>
00210 /// Create a new doughnut chart containing multiple concentric doughnuts.
00211 /// </summary>
00212 /// <param name="data">The data to plot.</param>
00213 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00214 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00215 /// <param name="innerRadius">The radius of the innermost doughnut "hole".  If this is <see
      cref="double.NaN"/>, it is determined automatically based on the number of concentric
      doughnuts.</param>
00216 /// <param name="margin">The spacing between consecutive doughnut rings, ranging from 0 to 1.</param>
```

```
00217 /// <param name="width">The width of the plot.</param>
00218 /// <param name="height">The height of the plot.</param>
00219 /// <param name="title">Title for the plot.</param>
00220 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00221 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00222 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00223 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00224             public static Plot StackedDoughnutChart(IReadOnlyList<IReadOnlyList<double» data, bool
      clockwise = false, double startAngle = 0, double innerRadius = double.NaN, double margin = 0.1, double
      width = 350, double height = 250,
00225                 string title = null,
00226                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00227                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00228                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00229             {
00230
00231                 if (dataPresentationAttributes == null)
00232                 {
00233                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill
      = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 0.5 },
00234                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill
      = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), LineWidth = 0.5 },
00235                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)),
      Fill = new SolidColourBrush(Colour.FromRgb(204, 121, 167)), LineWidth = 0.5 },
00236                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), Fill
      = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), LineWidth = 0.5 },
00237                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)),
      Fill = new SolidColourBrush(Colour.FromRgb(86, 180, 233)), LineWidth = 0.5 },
00238                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), Fill
      = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), LineWidth = 0.5 },
00239                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)),
      Fill = new SolidColourBrush(Colour.FromRgb(240, 228, 66)), LineWidth = 0.5 } };
00240                 }
00241
00242                 if (double.IsNaN(innerRadius))
00243                 {
00244                     innerRadius = Math.Max(0.1, Math.Pow(2, -data.Count));
00245                 }
00246
00247                 if (coordinateSystem == null)
00248                 {
00249                     double pieWidth = 2.5 + 2;
00250                     double pieHeight = 2.5 + 2;
00251
00252                     double aspectRatio = pieWidth / pieHeight;
00253                     double targetAspectRatio = width / height;
00254
00255                     if (targetAspectRatio > aspectRatio)
00256                     {
00257                         pieWidth = targetAspectRatio * pieHeight;
00258                     }
00259                     else
00260                     {
00261                         pieHeight = pieWidth / targetAspectRatio;
00262                     }
00263
00264                     coordinateSystem = new LinearCoordinateSystem2D(-1, pieWidth - 1, -pieHeight + 1,
      1, width, height);
00265                 }
00266
00267                 if (titlePresentationAttributes == null)
00268                 {
00269                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00270                 }
00271
00272                 double actualMargin = margin * (1 - innerRadius) / data.Count;
00273
00274                 Plot tbr = new Plot();
00275
00276                 for (int i = 0; i < data.Count; i++)
00277                 {
00278
00279                     Pie pie = new Pie(data[i], new double[] { 0, 0 }, new double[] { innerRadius + (1
      - innerRadius) * (i + 1) / data.Count - actualMargin * 0.5, innerRadius + (1 - innerRadius) * (i + 1)
      / data.Count - actualMargin * 0.5 }, coordinateSystem)
00280                     {
00281                         PresentationAttributes = dataPresentationAttributes,
```

```
00282                         Clockwise = clockwise,
00283                         StartAngle = startAngle,
00284                         InnerRadius = new double[] { innerRadius + (1 - innerRadius) * i / data.Count
      + actualMargin * 0.5, innerRadius + (1 - innerRadius) * i / data.Count + actualMargin * 0.5 }
00285                     };
00286
00287                     tbr.AddPlotElement(pie);
00288                 }
00289
00290                 Point top = coordinateSystem.ToPlotCoordinates(new double[] { 0, 1 });
00291                 double[] titleCentre = coordinateSystem.ToDataCoordinates(new Point(top.X, top.Y -
      10));
00292
00293                 TextLabel<IReadOnlyList<double> titleLabel = new
      TextLabel<IReadOnlyList<double>(title, titleCentre, coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00294                 tbr.AddPlotElement(titleLabel);
00295
00296
00297                 return tbr;
00298             }
00299         }
00300     }
00301 }
```

8.37 Plot.ScatterPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new scatter plot.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="width">The width of the plot.</param>
00033 /// <param name="height">The height of the plot.</param>
00034 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00035 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00036 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00037 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00038 /// <param name="xAxisTitle">Title for the X axis.</param>
00039 /// <param name="yAxisTitle">Title for the Y axis.</param>
00040 /// <param name="title">Title for the plot.</param>
00041 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00042 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00043 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00044 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00045 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00048             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<IReadOnlyList<double>> data,
      double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
```

```
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00059                 IReadOnlyList<double> pointSizes = null,
00060                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00061                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00062             {
00063                 if (axisPresentationAttributes == null)
00064                 {
00065                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00066                 }
00067 
00068                 if (gridPresentationAttributes == null)
00069                 {
00070                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00071                 }
00072 
00073                 if (axisLabelPresentationAttributes == null)
00074                 {
00075                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00076                 }
00077 
00078                 if (axisTitlePresentationAttributes == null)
00079                 {
00080                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00081                 }
00082 
00083                 if (dataPresentationAttributes == null)
00084                 {
00085                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00086                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00087                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00088                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00089                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
      180, 233)) },
00090                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00091                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00092                 }
00093 
00094                 if (dataPointElements == null)
00095                 {
00096                     dataPointElements = new PathDataPointElement[] { new PathDataPointElement() };
00097                 }
00098 
00099                 if (pointSizes == null)
00100                 {
00101                     pointSizes = new double[] { 2 };
00102                 }
00103 
00104                 List<IReadOnlyList<double» allData = data.Aggregate(new List<IReadOnlyList<double»(),
      (a, b) => { a.AddRange(b); return a; });
00105 
00106                 if (coordinateSystem == null)
00107                 {
00108                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00109                 }
00110 
00111                 if (titlePresentationAttributes == null)
00112                 {
00113                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00114                 }
00115 
00116                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00117 
00118                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00119                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
```

```
00120                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00121                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00122 
00123                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00124                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00125                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00126                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00127 
00128                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00129                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00130                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00131                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00132 
00133 
00134                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00135                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00136                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00137                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00138 
00139                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00140                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00141 
00142                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00143                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = axisArrowSize };
00144 
00145                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00146                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00147 
00148                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00149                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00150 
00151                 Graphics xLabelsSize = new Graphics();
00152                 xLabels.Plot(xLabelsSize);
00153                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00154 
00155                 Graphics yLabelsSize = new Graphics();
00156                 yLabels.Plot(yLabelsSize);
00157                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00158 
00159                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00160                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00161 
00162                 Plot tbr = new Plot();
00163                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00164 
00165                 for (int i = 0; i < data.Count; i++)
00166                 {
00167                     ScatterPoints<IReadOnlyList<double» points = new
```

```
      ScatterPoints<IReadOnlyList<double»(data[i], coordinateSystem) { PresentationAttributes =
      dataPresentationAttributes[i % dataPresentationAttributes.Count], Size = pointSizes[i %
      pointSizes.Count], DataPointElement = dataPointElements[i % dataPointElements.Count] };
00168                     tbr.AddPlotElement(points);
00169                 }
00170
00171
00172                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20 )), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00173
00174                 tbr.AddPlotElement(titleLabel);
00175
00176                 return tbr;
00177             }
00178
00179 /// <summary>
00180 /// Create a new scatter plot.
00181 /// </summary>
00182 /// <param name="data">The data to plot.</param>
00183 /// <param name="width">The width of the plot.</param>
00184 /// <param name="height">The height of the plot.</param>
00185 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00186 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00187 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00188 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00189 /// <param name="xAxisTitle">Title for the X axis.</param>
00190 /// <param name="yAxisTitle">Title for the Y axis.</param>
00191 /// <param name="title">Title for the plot.</param>
00192 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00193 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00194 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00195 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00196 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00197 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00198 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00199             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<double» data, double width =
      350, double height = 250,
00200                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00201                 double axisArrowSize = 3,
00202                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00203                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00204                 string xAxisTitle = null,
00205                 string yAxisTitle = null,
00206                 string title = null,
00207                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00208                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00209                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00210                 double pointSize = 2,
00211                 IDataPointElement dataPointElement = null,
00212                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00213             {
00214                 return ScatterPlot(new IReadOnlyList<IReadOnlyList<double»[] { data }, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes == null ?  null :  new
      PlotElementPresentationAttributes[] { dataPresentationAttributes }, new double[] { pointSize },
      dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement }, coordinateSystem);
00215             }
00216
00217 /// <summary>
00218 /// Create a new scatter plot.
00219 /// </summary>
00220 /// <param name="data">The data to plot.</param>
00221 /// <param name="width">The width of the plot.</param>
00222 /// <param name="height">The height of the plot.</param>
00223 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00224 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00225 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00226 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00227 /// <param name="xAxisTitle">Title for the X axis.</param>
00228 /// <param name="yAxisTitle">Title for the Y axis.</param>
00229 /// <param name="title">Title for the plot.</param>
00230 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00231 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00232 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00233 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00234 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00235 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00236 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00237             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<(double, double)» data, double
      width = 350, double height = 250,
00238                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00239                 double axisArrowSize = 3,
```

```
00240                PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00241                PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00242                string xAxisTitle = null,
00243                string yAxisTitle = null,
00244                string title = null,
00245                PlotElementPresentationAttributes titlePresentationAttributes = null,
00246                PlotElementPresentationAttributes gridPresentationAttributes = null,
00247                IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00248                IReadOnlyList<double> pointSizes = null,
00249                IReadOnlyList<IDataPointElement> dataPointElements = null,
00250                IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00251            {
00252                 return ScatterPlot((from el in data select (from el2 in el select new double[] {
      el2.Item1, el2.Item2 }).ToArray()).ToArray(), width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      dataPresentationAttributes, pointSizes, dataPointElements, coordinateSystem);
00253            }
00254
00255 /// <summary>
00256 /// Create a new scatter plot.
00257 /// </summary>
00258 /// <param name="data">The data to plot.</param>
00259 /// <param name="width">The width of the plot.</param>
00260 /// <param name="height">The height of the plot.</param>
00261 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00262 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00263 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00264 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00265 /// <param name="xAxisTitle">Title for the X axis.</param>
00266 /// <param name="yAxisTitle">Title for the Y axis.</param>
00267 /// <param name="title">Title for the plot.</param>
00268 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00269 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00270 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00271 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00272 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00273 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00274 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00275            public static Plot ScatterPlot(IReadOnlyList<(double, double)> data, double width = 350,
      double height = 250,
00276            PlotElementPresentationAttributes axisPresentationAttributes = null,
00277            double axisArrowSize = 3,
00278            PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00279            PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00280            string xAxisTitle = null,
00281            string yAxisTitle = null,
00282            string title = null,
00283            PlotElementPresentationAttributes titlePresentationAttributes = null,
00284            PlotElementPresentationAttributes gridPresentationAttributes = null,
00285            PlotElementPresentationAttributes dataPresentationAttributes = null,
00286            double pointSize = 2,
00287            IDataPointElement dataPointElement = null,
00288            IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00289            {
00290                return ScatterPlot((from el in data select new double[] { el.Item1, el.Item2
      }).ToArray(), width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes, pointSize,
      dataPointElement, coordinateSystem);
00291            }
00292
00293 /// <summary>
00294 /// Create a new scatter plot.
00295 /// </summary>
00296 /// <param name="data">The data to plot.</param>
00297 /// <param name="width">The width of the plot.</param>
00298 /// <param name="height">The height of the plot.</param>
00299 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00300 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00301 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00302 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00303 /// <param name="xAxisTitle">Title for the X axis.</param>
00304 /// <param name="yAxisTitle">Title for the Y axis.</param>
00305 /// <param name="title">Title for the plot.</param>
00306 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00307 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00308 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00309 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00310 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00311 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00312 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00313            public static Plot ScatterPlot(IReadOnlyList<Point> data, double width = 350, double
      height = 250,
00314                PlotElementPresentationAttributes axisPresentationAttributes = null,
```

```
00315                 double axisArrowSize = 3,
00316                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00317                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00318                 string xAxisTitle = null,
00319                 string yAxisTitle = null,
00320                 string title = null,
00321                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00322                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00323                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00324                 double pointSize = 2,
00325                 IDataPointElement dataPointElement = null,
00326                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00327             {
00328                 return ScatterPlot((from el in data select new double[] { el.X, el.Y }).ToArray(),
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, pointSize, dataPointElement,
      coordinateSystem);
00329             }
00330
00331 /// <summary>
00332 /// Create a new scatter plot.
00333 /// </summary>
00334 /// <param name="data">The data to plot.</param>
00335 /// <param name="width">The width of the plot.</param>
00336 /// <param name="height">The height of the plot.</param>
00337 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00338 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00339 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00340 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00341 /// <param name="xAxisTitle">Title for the X axis.</param>
00342 /// <param name="yAxisTitle">Title for the Y axis.</param>
00343 /// <param name="title">Title for the plot.</param>
00344 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00345 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00346 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00347 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00348 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00349 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00350 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00351             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<Point» data, double width =
      350, double height = 250,
00352                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00353                 double axisArrowSize = 3,
00354                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00355                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00356                 string xAxisTitle = null,
00357                 string yAxisTitle = null,
00358                 string title = null,
00359                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00360                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00361                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00362                 IReadOnlyList<double> pointSizes = null,
00363                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00364                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00365             {
00366                 return ScatterPlot((from el in data select (from el2 in el select new double[] {
      el2.X, el2.Y }).ToArray()).ToArray(), width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes, pointSizes,
      dataPointElements, coordinateSystem);
00367             }
00368         }
00369     }
00370 }
```

8.38 Plot.ViolinPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
```

```
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using MathNet.Numerics.Statistics;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025     partial class Plot
00026     {
00027         partial class Create
00028         {
00029 /// <summary>
00030 /// Create a new violin plot.
00031 /// </summary>
00032 /// <param name="data">The data to plot.</param>
00033 /// <param name="vertical">If this is <see langword="true"/> (the default), the violins are parallel
      to the Y axis.  If this is <see langword="false"/>, the violins are parallel to the X axis.</param>
00034 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
      this is <see langword="false"/>, it is not smoothed.</param>
00035 /// <param name="sides">Determines on which side(s) the violins are drawn.</param>
00036 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of each violin plot.</param>
00037 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the violins
      have the same width.  Otherwise, the width of each violin is proportional to the number of
      samples.</param>
00038 /// <param name="violinWidth">The width of the violins in data space coordinates.</param>
00039 /// <param name="boxWidth">The width of box plots as a proportion of the width of the violins.</param>
00040 /// <param name="spacing">The spacing between consecutive violins (ranging from 0 to 1).</param>
00041 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00042 /// even though the sampled values have different ranges.</param>
00043 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00044 /// even though the sampled values have different ranges.</param>
00045 /// <param name="width">The width of the plot.</param>
00046 /// <param name="height">The height of the plot.</param>
00047 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00048 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00049 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00050 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00051 /// <param name="xAxisTitle">Title for the X axis.</param>
00052 /// <param name="yAxisTitle">Title for the Y axis.</param>
00053 /// <param name="title">Title for the plot.</param>
00054 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00055 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00056 /// <param name="violinPresentationAttributes">Presentation attributes for the violins.</param>
00057 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00058 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00059 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00060 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00061 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00062             public static Plot ViolinPlot<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, bool
      proportionalWidth = false, bool smooth = true, Violin.ViolinSide sides = Violin.ViolinSide.Both, bool
      showBoxPlots = true, double violinWidth = 10, double boxWidth = 0.05, double spacing = 0.1, double?
      dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
      height = 250,
00063                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00064                 double axisArrowSize = 3,
00065                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00066                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00067                 string xAxisTitle = null,
00068                 string yAxisTitle = null,
00069                 string title = null,
00070                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00071                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00072                 IReadOnlyList<PlotElementPresentationAttributes> violinPresentationAttributes = null,
00073                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00074                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00075                 IDataPointElement medianSymbol = null,
00076                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00077             {
00078                 if (violinPresentationAttributes == null)
00079                 {
00080                     violinPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00081                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)),
      Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00082                                                                                 new
```

```
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
      Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00083                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
      Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00084                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00085                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00086                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00087                 }
00088
00089                 if (boxPresentationAttributes == null)
00090                 {
00091                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[violinPresentationAttributes.Count];
00092
00093                     for (int i = 0; i < violinPresentationAttributes.Count; i++)
00094                     {
00095                         temp[i] = new PlotElementPresentationAttributes() { Stroke =
      violinPresentationAttributes[i].Stroke, Fill = violinPresentationAttributes[i].Stroke };
00096                     }
00097
00098                     boxPresentationAttributes = temp;
00099                 }
00100
00101                 if (medianPresentationAttributes == null)
00102                 {
00103                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[violinPresentationAttributes.Count];
00104
00105                     for (int i = 0; i < violinPresentationAttributes.Count; i++)
00106                     {
00107                         temp[i] = new PlotElementPresentationAttributes() { Stroke = null, Fill =
      violinPresentationAttributes[i].Fill };
00108                     }
00109
00110                     medianPresentationAttributes = temp;
00111                 }
00112
00113                 if (medianSymbol == null)
00114                 {
00115                     medianSymbol = new PathDataPointElement();
00116                 }
00117
00118                 if (axisPresentationAttributes == null)
00119                 {
00120                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00121                 }
00122
00123                 if (gridPresentationAttributes == null)
00124                 {
00125                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00126                 }
00127
00128                 if (axisLabelPresentationAttributes == null)
00129                 {
00130                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00131                 }
00132
00133                 if (axisTitlePresentationAttributes == null)
00134                 {
00135                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00136                 }
00137
00138                 if (titlePresentationAttributes == null)
00139                 {
00140                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00141                 }
00142
00143                 double[] medians = new double[data.Count];
00144                 double[] firstQuartiles = new double[data.Count];
00145                 double[] thirdQuartiles = new double[data.Count];
00146                 double[] upperWhiskers = new double[data.Count];
00147                 double[] lowerWhiskers = new double[data.Count];
00148                 double[][] centredData = new double[data.Count][];
00149
```

```
00150                 double minData = double.MaxValue;
00151                 double maxData = double.MinValue;
00152 
00153                 for (int i = 0; i < data.Count; i++)
00154                 {
00155                     medians[i] = data[i].Item2.Median();
00156                     firstQuartiles[i] = data[i].Item2.LowerQuartile();
00157                     thirdQuartiles[i] = data[i].Item2.UpperQuartile();
00158 
00159 
00160                     upperWhiskers[i] = thirdQuartiles[i] + (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00161                     lowerWhiskers[i] = firstQuartiles[i] - (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00162 
00163                     double uW = double.MinValue;
00164                     double lW = double.MaxValue;
00165 
00166                     for (int j = 0; j < data[i].Item2.Count; j++)
00167                     {
00168                         if (data[i].Item2[j] <= upperWhiskers[i])
00169                         {
00170                             uW = Math.Max(uW, data[i].Item2[j]);
00171                         }
00172 
00173                         if (data[i].Item2[j] >= lowerWhiskers[i])
00174                         {
00175                             lW = Math.Min(lW, data[i].Item2[j]);
00176                         }
00177 
00178                         minData = Math.Min(minData, data[i].Item2[j]);
00179                         maxData = Math.Max(maxData, data[i].Item2[j]);
00180                     }
00181 
00182                     upperWhiskers[i] = uW;
00183                     lowerWhiskers[i] = lW;
00184 
00185                     centredData[i] = new double[data[i].Item2.Count];
00186 
00187                     for (int j = 0; j < data[i].Item2.Count; j++)
00188                     {
00189                         centredData[i][j] = data[i].Item2[j] - medians[i];
00190                     }
00191                 }
00192 
00193                 double minX, maxX, minY, maxY;
00194 
00195                 if (vertical)
00196                 {
00197                     minX = 0;
00198                     maxX = violinWidth * (data.Count + spacing * (data.Count - 1));
00199 
00200                     minY = minData;
00201                     maxY = maxData;
00202 
00203                     if (dataRangeMin != null)
00204                     {
00205                         minY = dataRangeMin.Value;
00206                     }
00207 
00208                     if (dataRangeMax != null)
00209                     {
00210                         maxY = dataRangeMax.Value;
00211                     }
00212                 }
00213                 else
00214                 {
00215                     minY = 0;
00216                     maxY = violinWidth * (data.Count + spacing * (data.Count - 1));
00217 
00218                     minX = minData;
00219                     maxX = maxData;
00220 
00221                     if (dataRangeMin != null)
00222                     {
00223                         minX = dataRangeMin.Value;
00224                     }
00225 
00226                     if (dataRangeMax != null)
00227                     {
00228                         maxX = dataRangeMax.Value;
00229                     }
00230                 }
00231 
00232 
00233                 if (coordinateSystem == null)
00234                 {
```

```
00235                   coordinateSystem = new LinearCoordinateSystem2D(minX, maxX, minY, maxY, width,
      height);
00236               }
00237
00238               Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00239               Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00240               Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00241               Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00242
00243               double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00244               double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00245               double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00246               double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00247
00248               double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00249               double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00250               double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00251               double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00252
00253
00254               double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00255               double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00256               double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00257               double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00258
00259               ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft, marginBottomRight,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = vertical ?  0 :
      axisArrowSize };
00260               ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft, marginTopLeft,
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, ArrowSize = vertical ?
      axisArrowSize :  0 };
00261
00262               IPlotElement xTicks;
00263               IPlotElement yTicks;
00264
00265               IPlotElement xLabels;
00266               IPlotElement yLabels;
00267
00268               if (vertical)
00269               {
00270                   xTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { (el * (spacing + 1) + 0.5) * violinWidth, p3[1] }),
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement =
      new PathDataPointElement(new GraphicsPath().MoveTo(0, -3).LineTo(0, 3)) };
00271                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00272
00273                   xLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { (el * (spacing + 1) + 0.5) * violinWidth, p3[1] }),
      coordinateSystem) { Alignment = TextAnchors.Center, Baseline = TextBaselines.Top, Label = (i, _) =>
      data[i].Item1, PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new
      Point(0, 10) };
00274                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00275               }
00276               else
00277               {
00278                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00279
00280                   yTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * violinWidth }),
```

```
coordinateSystem) { PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new PathDataPointElement(new GraphicsPath().MoveTo(-3, 0).LineTo(3, 0)) };
00281
00282                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) { PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline = TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00283                   yLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0, data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * violinWidth }), coordinateSystem) { Alignment = TextAnchors.Right, Baseline = TextBaselines.Middle, Label = (i, _) => data[i].Item1, PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(-10, 0) };
00284               }
00285
00286               Graphics xLabelsSize = new Graphics();
00287               xLabels.Plot(xLabelsSize);
00288               double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00289
00290               Graphics yLabelsSize = new Graphics();
00291               yLabels.Plot(yLabelsSize);
00292               double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00293
00294               ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft, marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20, Alignment = TextAnchors.Center };
00295               ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft, marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth, Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00296
00297               Plot plot = new Plot();
00298
00299               if (vertical)
00300               {
00301                   Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5, PresentationAttributes = gridPresentationAttributes };
00302                   plot.AddPlotElement(yGrid);
00303               }
00304               else
00305               {
00306                   Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5, PresentationAttributes = gridPresentationAttributes };
00307                   plot.AddPlotElement(xGrid);
00308               }
00309
00310               plot.AddPlotElements(xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle, yTitle);
00311
00312               double maxCount = (from el in data select el.Item2.Count).Max();
00313
00314               for (int i = 0; i < data.Count; i++)
00315               {
00316                   if (vertical)
00317                   {
00318                       Violin violin = new Violin(new double[] { (i * (spacing + 1) + 0.5) * violinWidth, medians[i] }, new double[] { 0, 1 }, centredData[i], coordinateSystem)
00319                       {
00320                           PresentationAttributes = violinPresentationAttributes[i % boxPresentationAttributes.Count],
00321                           Width = proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count / maxCount) :  (violinWidth * 0.5),
00322                           Smooth = smooth,
00323                           Sides = sides
00324                       };
00325                       plot.AddPlotElement(violin);
00326
00327                       if (showBoxPlots)
00328                       {
00329                           BoxPlot box = new BoxPlot(new double[] { (i * (spacing + 1) + 0.5) * violinWidth, medians[i] }, new double[] { 0, 1 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i], thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00330                           {
00331                               BoxPresentationAttributes = boxPresentationAttributes[i % boxPresentationAttributes.Count],
00332                               WhiskersPresentationAttributes = boxPresentationAttributes[i % boxPresentationAttributes.Count],
00333                               Width = (proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count / maxCount) :  (violinWidth * 0.5)) * boxWidth,
00334                               NotchSize = 0,
00335                               WhiskerWidth = 0,
00336                               CentreSymbolPresentationAttributes = medianPresentationAttributes[i % medianPresentationAttributes.Count],
00337                               CentreSymbol = medianSymbol
00338                           };
00339
00340                           plot.AddPlotElement(box);
00341                       }
00342                   }
00343                   else
00344                   {
```

```
00345                         Violin violin = new Violin(new double[] { medians[i], (i * (spacing + 1) +
      0.5) * violinWidth }, new double[] { 1, 0 }, centredData[i], coordinateSystem)
00346                         {
00347                             PresentationAttributes = violinPresentationAttributes[i %
      boxPresentationAttributes.Count],
00348                             Width = proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (violinWidth * 0.5),
00349                             Smooth = smooth,
00350                             Sides = sides
00351                         };
00352                         plot.AddPlotElement(violin);
00353 
00354                         if (showBoxPlots)
00355                         {
00356                             BoxPlot box = new BoxPlot(new double[] { medians[i], (i * (spacing + 1) +
      0.5) * violinWidth }, new double[] { 1, 0 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] -
      medians[i], thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00357                             {
00358                                 BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00359                                 WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00360                                 Width = (proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count
      / maxCount) :  (violinWidth * 0.5)) * boxWidth,
00361                                 NotchSize = 0,
00362                                 WhiskerWidth = 0,
00363                                 CentreSymbolPresentationAttributes = medianPresentationAttributes[i %
      medianPresentationAttributes.Count],
00364                                 CentreSymbol = medianSymbol
00365                             };
00366 
00367                             plot.AddPlotElement(box);
00368                         }
00369                     }
00370                 }
00371 
00372                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, coordinateSystem.ToDataCoordinates(new Point((topLeft.X +
      topRight.X) * 0.5, (topLeft.Y + topRight.Y) * 0.5 - 20)), coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00373 
00374                 plot.AddPlotElement(titleLabel);
00375 
00376                 return plot;
00377             }
00378 
00379 /// <summary>
00380 /// Create a new violin plot.
00381 /// </summary>
00382 /// <param name="data">The data to plot.</param>
00383 /// <param name="vertical">If this is <see langword="true"/> (the default), the violins are parallel
      to the Y axis.  If this is <see langword="false"/>, the violins are parallel to the X axis.</param>
00384 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
      this is <see langword="false"/>, it is not smoothed.</param>
00385 /// <param name="sides">Determines on which side(s) the violins are drawn.</param>
00386 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of each violin plot.</param>
00387 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the violins
      have the same width.  Otherwise, the width of each violin is proportional to the number of
      samples.</param>
00388 /// <param name="violinWidth">The width of the violins in data space coordinates.</param>
00389 /// <param name="boxWidth">The width of box plots as a proportion of the width of the violins.</param>
00390 /// <param name="spacing">The spacing between consecutive violins (ranging from 0 to 1).</param>
00391 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00392 /// even though the sampled values have different ranges.</param>
00393 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00394 /// even though the sampled values have different ranges.</param>
00395 /// <param name="width">The width of the plot.</param>
00396 /// <param name="height">The height of the plot.</param>
00397 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00398 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00399 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00400 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00401 /// <param name="xAxisTitle">Title for the X axis.</param>
00402 /// <param name="yAxisTitle">Title for the Y axis.</param>
00403 /// <param name="title">Title for the plot.</param>
00404 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00405 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00406 /// <param name="violinPresentationAttributes">Presentation attributes for the violins.</param>
00407 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00408 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00409 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
```

```
00410 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00411 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00412             public static Plot ViolinPlot(IReadOnlyList<IReadOnlyList<double» data, bool
      proportionalWidth = false, bool smooth = true, Violin.ViolinSide sides = Violin.ViolinSide.Both, bool
      showBoxPlots = true, double violinWidth = 10, double boxWidth = 0.05, double spacing = 0.1, double?
      dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
      height = 250,
00413                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00414                 double axisArrowSize = 3,
00415                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00416                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00417                 string xAxisTitle = null,
00418                 string yAxisTitle = null,
00419                 string title = null,
00420                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00421                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00422                 IReadOnlyList<PlotElementPresentationAttributes> violinPresentationAttributes = null,
00423                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00424                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00425                 IDataPointElement medianSymbol = null,
00426                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00427             {
00428                 return ViolinPlot((from el in data select ((string)null, el)).ToArray(),
      proportionalWidth, smooth, sides, showBoxPlots, violinWidth, boxWidth, spacing, dataRangeMin,
      dataRangeMax, vertical, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, violinPresentationAttributes,
      boxPresentationAttributes, medianPresentationAttributes, medianSymbol, coordinateSystem);
00429             }
00430
00431 /// <summary>
00432 /// Create a new violin plot.
00433 /// </summary>
00434 /// <param name="data">The data to plot.</param>
00435 /// <param name="vertical">If this is <see langword="true"/> (the default), the violin is parallel to
      the Y axis.  If this is <see langword="false"/>, the violin is parallel to the X axis.</param>
00436 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
      this is <see langword="false"/>, it is not smoothed.</param>
00437 /// <param name="sides">Determines on which side(s) the violin is drawn.</param>
00438 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of the violin plot.</param>
00439 /// <param name="violinWidth">The width of the violin in data space coordinates.</param>
00440 /// <param name="boxWidth">The width of box plot as a proportion of the width of the violin.</param>
00441 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00442 /// even though the sampled values have different ranges.</param>
00443 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00444 /// even though the sampled values have different ranges.</param>
00445 /// <param name="width">The width of the plot.</param>
00446 /// <param name="height">The height of the plot.</param>
00447 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00448 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00449 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00450 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00451 /// <param name="xAxisTitle">Title for the X axis.</param>
00452 /// <param name="yAxisTitle">Title for the Y axis.</param>
00453 /// <param name="title">Title for the plot.</param>
00454 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00455 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00456 /// <param name="violinPresentationAttributes">Presentation attributes for the violin.</param>
00457 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00458 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00459 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00460 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00461 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00462             public static Plot ViolinPlot(IReadOnlyList<double> data, bool smooth = true,
      Violin.ViolinSide sides = Violin.ViolinSide.Both, bool showBoxPlots = true, double violinWidth = 10,
      double boxWidth = 0.05, double?  dataRangeMin = null, double?  dataRangeMax = null, bool vertical =
      true, double width = 350, double height = 250,
00463                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00464                 double axisArrowSize = 3,
00465                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00466                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00467                 string xAxisTitle = null,
00468                 string yAxisTitle = null,
00469                 string title = null,
00470                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00471                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00472                 PlotElementPresentationAttributes violinPresentationAttributes = null,
00473                 PlotElementPresentationAttributes boxPresentationAttributes = null,
00474                 PlotElementPresentationAttributes medianPresentationAttributes = null,
```

```
00475                 IDataPointElement medianSymbol = null,
00476                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00477             {
00478                 return ViolinPlot(new (string, IReadOnlyList<double>)[] { (null, data) }, false,
      smooth, sides, showBoxPlots, violinWidth, boxWidth, 0, dataRangeMin, dataRangeMax, vertical, width,
      height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, violinPresentationAttributes == null ? null : new
      PlotElementPresentationAttributes[] { violinPresentationAttributes }, boxPresentationAttributes ==
      null ? null : new PlotElementPresentationAttributes[] { boxPresentationAttributes },
      medianPresentationAttributes == null ? null : new PlotElementPresentationAttributes[] {
      medianPresentationAttributes }, medianSymbol, coordinateSystem);
00479             }
00480         }
00481     }
00482 }
```

8.39 Trendlines.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program. If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using MathNet.Numerics;
00019 using MathNet.Numerics.LinearAlgebra;
00020 using MathNet.Numerics.LinearAlgebra.Double;
00021 using MathNet.Numerics.LinearRegression;
00022 using System;
00023 using System.Collections.Generic;
00024 using System.Linq;
00025
00026 namespace VectSharp.Plots
00027 {
00028 /// <summary>
00029 /// A plot element that draws a linear trendline with equation <c>y = a * x + b</c>.
00030 /// </summary>
00031     public class LinearTrendLine : IPlotElement
00032     {
00033 /// <summary>
00034 /// The slope of the trendline (a).
00035 /// </summary>
00036         public double Slope { get; set; }
00037
00038 /// <summary>
00039 /// The intercept of the trendline (b).
00040 /// </summary>
00041         public double Intercept { get; set; }
00042
00043 /// <summary>
00044 /// The minimum X value for which the trendline is plotted.
00045 /// </summary>
00046         public double MinX { get; set; }
00047
00048 /// <summary>
00049 /// The minimum Y value for which the trendline is plotted.
00050 /// </summary>
00051         public double MinY { get; set; }
00052
00053 /// <summary>
00054 /// The maximum X value for which the trendline is plotted.
00055 /// </summary>
00056         public double MaxX { get; set; }
00057
00058 /// <summary>
00059 /// The maximum Y value for which the trendline is plotted.
00060 /// </summary>
00061         public double MaxY { get; set; }
00062
00063 /// <summary>
00064 /// Presentation attributes for the trendline.
```

```
00065 /// </summary>
00066         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00067
00068 /// <summary>
00069 /// A tag to identify the trendline in the plot.
00070 /// </summary>
00071         public string Tag { get; set; }
00072
00073 /// <summary>
00074 /// The coordinate system used to transform the points from data space to plot space.
00075 /// </summary>
00076         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00077         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00078
00079 /// <summary>
00080 /// Create a new <see cref="LinearTrendLine"/> instance, specifying the equation parameters.
00081 /// </summary>
00082 /// <param name="slope">The slope of the trendline (a).</param>
00083 /// <param name="intercept">The intercept of the trendline (b).</param>
00084 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00085 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00086 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00087 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00088 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00089         public LinearTrendLine(double slope, double intercept, double minX, double minY, double maxX,
      double maxY, IContinuousCoordinateSystem coordinateSystem)
00090         {
00091             Slope = slope;
00092             Intercept = intercept;
00093             MinX = minX;
00094             MinY = minY;
00095             MaxX = maxX;
00096             MaxY = maxY;
00097             CoordinateSystem = coordinateSystem;
00098         }
00099
00100 /// <summary>
00101 /// Create a new <see cref="LinearTrendLine"/> instance, determining the equation parameters by
      running a regression.
00102 /// </summary>
00103 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00104 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00105 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00106         public LinearTrendLine(IReadOnlyList<IReadOnlyList<double» data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null)
00107         {
00108             MinX = double.MaxValue;
00109             MinY = double.MaxValue;
00110             MaxX = double.MinValue;
00111             MaxY = double.MinValue;
00112
00113             double[] x = new double[data.Count];
00114             double[] y = new double[data.Count];
00115
00116             for (int i = 0; i < data.Count; i++)
00117             {
00118                 MinX = Math.Min(MinX, data[i][0]);
00119                 MinY = Math.Min(MinY, data[i][1]);
00120                 MaxX = Math.Max(MaxX, data[i][0]);
00121                 MaxY = Math.Max(MaxY, data[i][1]);
00122
00123                 x[i] = data[i][0];
00124
00125                 if (fixedIntercept == null)
00126                 {
00127                     y[i] = data[i][1];
00128                 }
00129                 else
00130                 {
00131                     y[i] = data[i][1] - fixedIntercept.Value;
00132                 }
00133             }
00134
00135             if (fixedIntercept == null)
00136             {
00137                 (Intercept, Slope) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00138             }
00139             else
00140             {
00141                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00142                 Intercept = fixedIntercept.Value;
00143             }
```

```
00144
00145             this.CoordinateSystem = coordinateSystem;
00146         }
00147
00148 /// <summary>
00149 /// Create a new <see cref="LinearTrendLine"/> instance, determining the equation parameters by
      running a regression.
00150 /// </summary>
00151 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00152 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00153 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00154         public LinearTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] {
      el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
00155
00156 /// <inheritdoc/>
00157         public void Plot(Graphics target)
00158         {
00159             double y0 = Slope * MinX + Intercept;
00160             double x0 = (MinY - Intercept) / Slope;
00161
00162             double y1 = Slope * MaxX + Intercept;
00163             double x1 = (MaxY - Intercept) / Slope;
00164
00165             HashSet<(double, double)> points = new HashSet<(double, double)>();
00166
00167             if (y0 >= MinY && y0 <= MaxY)
00168             {
00169                 points.Add((MinX, y0));
00170             }
00171
00172             if (x0 >= MinX && x0 <= MaxX)
00173             {
00174                 points.Add((x0, MinY));
00175             }
00176
00177             if (y1 >= MinY && y1 <= MaxY)
00178             {
00179                 points.Add((MaxX, y1));
00180             }
00181
00182             if (x1 >= MinX && x1 <= MaxX)
00183             {
00184                 points.Add((x1, MaxY));
00185             }
00186
00187             if (points.Count == 2)
00188             {
00189                 double[][] uniquePoints = new double[points.Count][];
00190
00191                 int index = 0;
00192
00193                 foreach ((double x, double y) in points)
00194                 {
00195                     uniquePoints[index] = new double[] { x, y };
00196                     index++;
00197                 }
00198
00199                 GraphicsPath path;
00200
00201                 if (CoordinateSystem.IsLinear || CoordinateSystem.IsDirectionStraight(new double[] {
      uniquePoints[1][0] - uniquePoints[0][0], uniquePoints[1][1] - uniquePoints[0][1] }))
00202                 {
00203                     Point p1 = CoordinateSystem.ToPlotCoordinates(uniquePoints[0]);
00204                     Point p2 = CoordinateSystem.ToPlotCoordinates(uniquePoints[1]);
00205
00206                     path = new GraphicsPath().MoveTo(p1).LineTo(p2);
00207                 }
00208                 else
00209                 {
00210                     double[] direction = new double[] { uniquePoints[1][0] - uniquePoints[0][0],
      uniquePoints[1][1] - uniquePoints[0][1] };
00211                     double mod = Math.Sqrt(direction[0] * direction[0] + direction[1] * direction[1]);
00212                     double[] increment = new double[] { direction[0] / mod *
      Math.Min(CoordinateSystem.Resolution[0], CoordinateSystem.Resolution[1]), direction[1] / mod *
      Math.Min(CoordinateSystem.Resolution[0], CoordinateSystem.Resolution[1]) };
00213
00214                     int count = (int)Math.Ceiling(Math.Max(direction[0] / increment[0], direction[1] /
      increment[1]));
00215
00216                     path = new GraphicsPath();
00217
00218                     for (int i = 0; i <= count; i++)
00219                     {
00220                         double[] pt = new double[] { uniquePoints[0][0] + increment[0] * i,
```

```
    uniquePoints[0][1] + increment[1] * i };
00221
00222                         if (i == 0)
00223                         {
00224                             path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00225                         }
00226                         else
00227                         {
00228                             path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00229                         }
00230                     }
00231                 }
00232
00233                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00234                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00235                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00236                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00237
00238                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
    bottomRight.X));
00239                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
    bottomRight.Y));
00240                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
    bottomRight.X));
00241                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
    bottomRight.Y));
00242
00243                 target.Save();
00244                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00245
00246                 target.StrokePath(path, PresentationAttributes.Stroke,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
    PresentationAttributes.LineDash, Tag);
00247                 target.Restore();
00248             }
00249
00250         }
00251     }
00252
00253 /// <summary>
00254 /// A plot element that draws an exponential trendline with equation <c>y = b * Exp(a * x)</c>.
00255 /// </summary>
00256     public class ExponentialTrendLine :  IPlotElement
00257     {
00258 /// <summary>
00259 /// The slope of the trendline (a).
00260 /// </summary>
00261         public double Slope { get; set; }
00262
00263 /// <summary>
00264 /// The intercept of the trendline (b).
00265 /// </summary>
00266         public double Intercept { get; set; }
00267
00268 /// <summary>
00269 /// The minimum X value for which the trendline is plotted.
00270 /// </summary>
00271         public double MinX { get; set; }
00272
00273 /// <summary>
00274 /// The minimum Y value for which the trendline is plotted.
00275 /// </summary>
00276         public double MinY { get; set; }
00277
00278 /// <summary>
00279 /// The maximum X value for which the trendline is plotted.
00280 /// </summary>
00281         public double MaxX { get; set; }
00282
00283 /// <summary>
00284 /// The maximum Y value for which the trendline is plotted.
00285 /// </summary>
00286         public double MaxY { get; set; }
00287
00288 /// <summary>
00289 /// Presentation attributes for the trendline.
00290 /// </summary>
00291         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
    PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
    180, 180) };
00292
00293 /// <summary>
00294 /// A tag to identify the trendline in the plot.
00295 /// </summary>
00296         public string Tag { get; set; }
00297
00298 /// <summary>
```

```
00299 /// The coordinate system used to transform the points from data space to plot space.
00300 /// </summary>
00301         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00302         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00303
00304 /// <summary>
00305 /// Create a new <see cref="ExponentialTrendLine"/> instance, specifying the equation parameters.
00306 /// </summary>
00307 /// <param name="slope">The slope of the trendline (a).</param>
00308 /// <param name="intercept">The intercept of the trendline (b).</param>
00309 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00310 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00311 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00312 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00313 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00314         public ExponentialTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00315         {
00316             Slope = slope;
00317             Intercept = intercept;
00318             MinX = minX;
00319             MinY = minY;
00320             MaxX = maxX;
00321             MaxY = maxY;
00322             CoordinateSystem = coordinateSystem;
00323         }
00324
00325 /// <summary>
00326 /// Create a new <see cref="ExponentialTrendLine"/> instance, determining the equation parameters by
      running a regression.
00327 /// </summary>
00328 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00329 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00330 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00331         public ExponentialTrendLine(IReadOnlyList<IReadOnlyList<double>> data,
      IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null)
00332         {
00333             if (fixedIntercept <= 0)
00334             {
00335                 throw new ArgumentOutOfRangeException(nameof(fixedIntercept), fixedIntercept, "The
      intercept must be greater than 0!");
00336             }
00337
00338             MinX = double.MaxValue;
00339             MinY = double.MaxValue;
00340             MaxX = double.MinValue;
00341             MaxY = double.MinValue;
00342
00343             double[] x = new double[data.Count];
00344             double[] y = new double[data.Count];
00345
00346             double shift = 0;
00347
00348             if (fixedIntercept != null)
00349             {
00350                 shift = Math.Log(fixedIntercept.Value);
00351             }
00352
00353             for (int i = 0; i < data.Count; i++)
00354             {
00355                 MinX = Math.Min(MinX, data[i][0]);
00356                 MinY = Math.Min(MinY, data[i][1]);
00357                 MaxX = Math.Max(MaxX, data[i][0]);
00358                 MaxY = Math.Max(MaxY, data[i][1]);
00359
00360                 x[i] = data[i][0];
00361                 y[i] = Math.Log(data[i][1]) - shift;
00362             }
00363
00364             if (fixedIntercept == null)
00365             {
00366                 (double k, double b) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00367                 Intercept = Math.Exp(k);
00368                 Slope = b;
00369             }
00370             else
00371             {
00372                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00373                 Intercept = fixedIntercept.Value;
00374             }
00375
00376             this.CoordinateSystem = coordinateSystem;
00377         }
00378
```

```
00379 /// <summary>
00380 /// Create a new <see cref="ExponentialTrendLine"/> instance, determining the equation parameters by
      running a regression.
00381 /// </summary>
00382 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00383 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00384 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00385         public ExponentialTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] {
      el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
00386
00387 /// <inheritdoc/>
00388         public void Plot(Graphics target)
00389         {
00390             double y0 = Math.Exp(Slope * MinX) * Intercept;
00391             double x0 = (Math.Log(MinY) - Math.Log(Intercept)) / Slope;
00392
00393             double y1 = Math.Exp(Slope * MaxX) * Intercept;
00394             double x1 = (Math.Log(MaxY) - Math.Log(Intercept)) / Slope;
00395
00396             HashSet<(double, double)> points = new HashSet<(double, double)>();
00397
00398             if (y0 >= MinY && y0 <= MaxY)
00399             {
00400                 points.Add((MinX, y0));
00401             }
00402
00403             if (x0 >= MinX && x0 <= MaxX)
00404             {
00405                 points.Add((x0, MinY));
00406             }
00407
00408             if (y1 >= MinY && y1 <= MaxY)
00409             {
00410                 points.Add((MaxX, y1));
00411             }
00412
00413             if (x1 >= MinX && x1 <= MaxX)
00414             {
00415                 points.Add((x1, MaxY));
00416             }
00417
00418             if (points.Count == 2)
00419             {
00420                 double[][] uniquePoints = new double[points.Count][];
00421
00422                 int index = 0;
00423
00424                 foreach ((double x, double y) in points)
00425                 {
00426                     uniquePoints[index] = new double[] { x, y };
00427                     index++;
00428                 }
00429
00430                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
00431                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
00432
00433                 int count = (int)Math.Ceiling(Math.Max((endX - startX) /
      CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) /
      CoordinateSystem.Resolution[1]));
00434
00435                 GraphicsPath path = new GraphicsPath();
00436
00437                 for (int i = 0; i <= count; i++)
00438                 {
00439                     double[] pt = new double[] { startX + (endX - startX) / count * i, Math.Exp(Slope
      * (startX + (endX - startX) / count * i)) * Intercept };
00440
00441                     if (i == 0)
00442                     {
00443                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00444                     }
00445                     else
00446                     {
00447                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00448                     }
00449                 }
00450
00451                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00452                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00453                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00454                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00455
00456                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
```

```
00457                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00458                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00459                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00460
00461                 target.Save();
00462                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00463
00464                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, Tag);
00465                 target.Restore();
00466             }
00467
00468         }
00469     }
00470
00471 /// <summary>
00472 /// A plot element that draws a logarithmic trendline with equation <c>y = a * Ln(x) + b</c>.
00473 /// </summary>
00474     public class LogarithmicTrendLine :  IPlotElement
00475     {
00476 /// <summary>
00477 /// The slope of the trendline (a).
00478 /// </summary>
00479         public double Slope { get; set; }
00480
00481 /// <summary>
00482 /// The intercept of the trendline (b).
00483 /// </summary>
00484         public double Intercept { get; set; }
00485
00486 /// <summary>
00487 /// The minimum X value for which the trendline is plotted.
00488 /// </summary>
00489         public double MinX { get; set; }
00490
00491 /// <summary>
00492 /// The minimum Y value for which the trendline is plotted.
00493 /// </summary>
00494         public double MinY { get; set; }
00495
00496 /// <summary>
00497 /// The maximum X value for which the trendline is plotted.
00498 /// </summary>
00499         public double MaxX { get; set; }
00500
00501 /// <summary>
00502 /// The maximum Y value for which the trendline is plotted.
00503 /// </summary>
00504         public double MaxY { get; set; }
00505
00506 /// <summary>
00507 /// Presentation attributes for the trendline.
00508 /// </summary>
00509         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00510
00511 /// <summary>
00512 /// A tag to identify the trendline in the plot.
00513 /// </summary>
00514         public string Tag { get; set; }
00515
00516 /// <summary>
00517 /// The coordinate system used to transform the points from data space to plot space.
00518 /// </summary>
00519         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00520         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00521
00522 /// <summary>
00523 /// Create a new <see cref="LogarithmicTrendLine"/> instance, specifying the equation parameters.
00524 /// </summary>
00525 /// <param name="slope">The slope of the trendline (a).</param>
00526 /// <param name="intercept">The intercept of the trendline (b).</param>
00527 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00528 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00529 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00530 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00531 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00532         public LogarithmicTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00533         {
00534             Slope = slope;
```

```
00535             Intercept = intercept;
00536             MinX = minX;
00537             MinY = minY;
00538             MaxX = maxX;
00539             MaxY = maxY;
00540             CoordinateSystem = coordinateSystem;
00541         }
00542 
00543 /// <summary>
00544 /// Create a new <see cref="LogarithmicTrendLine"/> instance, determining the equation parameters by running a regression.
00545 /// </summary>
00546 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00547 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00548 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.</param>
00549         public LogarithmicTrendLine(IReadOnlyList<IReadOnlyList<double» data, IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null)
00550         {
00551             MinX = double.MaxValue;
00552             MinY = double.MaxValue;
00553             MaxX = double.MinValue;
00554             MaxY = double.MinValue;
00555 
00556             double[] x = new double[data.Count];
00557             double[] y = new double[data.Count];
00558 
00559             double shift = 0;
00560 
00561             if (fixedIntercept != null)
00562             {
00563                 shift = fixedIntercept.Value;
00564             }
00565 
00566             for (int i = 0; i < data.Count; i++)
00567             {
00568                 MinX = Math.Min(MinX, data[i][0]);
00569                 MinY = Math.Min(MinY, data[i][1]);
00570                 MaxX = Math.Max(MaxX, data[i][0]);
00571                 MaxY = Math.Max(MaxY, data[i][1]);
00572 
00573                 x[i] = Math.Log(data[i][0]);
00574                 y[i] = data[i][1] - shift;
00575             }
00576 
00577             if (fixedIntercept == null)
00578             {
00579                 (Intercept, Slope) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00580             }
00581             else
00582             {
00583                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00584                 Intercept = fixedIntercept.Value;
00585             }
00586 
00587             this.CoordinateSystem = coordinateSystem;
00588         }
00589 
00590 /// <summary>
00591 /// Create a new <see cref="LogarithmicTrendLine"/> instance, determining the equation parameters by running a regression.
00592 /// </summary>
00593 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00594 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00595 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.</param>
00596         public LogarithmicTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] { el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
00597 
00598 /// <inheritdoc/>
00599         public void Plot(Graphics target)
00600         {
00601             double y0 = Slope * Math.Log(MinX) + Intercept;
00602             double x0 = Math.Exp((MinY - Intercept) / Slope);
00603 
00604             double y1 = Slope * Math.Log(MaxX) + Intercept;
00605             double x1 = Math.Exp((MaxY - Intercept) / Slope);
00606 
00607             HashSet<(double, double)> points = new HashSet<(double, double)>();
00608 
00609             if (y0 >= MinY && y0 <= MaxY)
00610             {
00611                 points.Add((MinX, y0));
00612             }
```

```
00613
00614             if (x0 >= MinX && x0 <= MaxX)
00615             {
00616                 points.Add((x0, MinY));
00617             }
00618
00619             if (y1 >= MinY && y1 <= MaxY)
00620             {
00621                 points.Add((MaxX, y1));
00622             }
00623
00624             if (x1 >= MinX && x1 <= MaxX)
00625             {
00626                 points.Add((x1, MaxY));
00627             }
00628
00629             if (points.Count == 2)
00630             {
00631                 double[][] uniquePoints = new double[points.Count][];
00632
00633                 int index = 0;
00634
00635                 foreach ((double x, double y) in points)
00636                 {
00637                     uniquePoints[index] = new double[] { x, y };
00638                     index++;
00639                 }
00640
00641                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
00642                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
00643
00644                 int count = (int)Math.Ceiling(Math.Max((endX - startX) /
    CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) /
    CoordinateSystem.Resolution[1]));
00645
00646                 GraphicsPath path = new GraphicsPath();
00647
00648                 for (int i = 0; i <= count; i++)
00649                 {
00650                     double[] pt = new double[] { startX + (endX - startX) / count * i, Slope *
    Math.Log(startX + (endX - startX) / count * i) + Intercept };
00651
00652                     if (i == 0)
00653                     {
00654                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00655                     }
00656                     else
00657                     {
00658                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00659                     }
00660                 }
00661
00662                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00663                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00664                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00665                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00666
00667                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
    bottomRight.X));
00668                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
    bottomRight.Y));
00669                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
    bottomRight.X));
00670                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
    bottomRight.Y));
00671
00672                 target.Save();
00673                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00674
00675                 target.StrokePath(path, PresentationAttributes.Stroke,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
    PresentationAttributes.LineDash, Tag);
00676                 target.Restore();
00677             }
00678
00679         }
00680     }
00681
00682 /// <summary>
00683 /// A plot element that draws a polynomial trendline with equation <c>y = a0 + a1 * x + a2 * x^2 + ...
    + aN * x^N</c>.
00684 /// </summary>
00685     public class PolynomialTrendLine :  IPlotElement
00686     {
00687 /// <summary>
00688 /// The coefficients (<c>a0 ...  aN</c>).
00689 /// </summary>
```

```
00690         public double[] Coefficients { get; set; }
00691 /// <summary>
00692 /// The minimum X value for which the trendline is plotted.
00693 /// </summary>
00694         public double MinX { get; set; }
00695
00696 /// <summary>
00697 /// The minimum Y value for which the trendline is plotted.
00698 /// </summary>
00699         public double MinY { get; set; }
00700
00701 /// <summary>
00702 /// The maximum X value for which the trendline is plotted.
00703 /// </summary>
00704         public double MaxX { get; set; }
00705
00706 /// <summary>
00707 /// The maximum Y value for which the trendline is plotted.
00708 /// </summary>
00709         public double MaxY { get; set; }
00710
00711 /// <summary>
00712 /// Presentation attributes for the trendline.
00713 /// </summary>
00714         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00715
00716 /// <summary>
00717 /// A tag to identify the trendline in the plot.
00718 /// </summary>
00719         public string Tag { get; set; }
00720
00721 /// <summary>
00722 /// The coordinate system used to transform the points from data space to plot space.
00723 /// </summary>
00724         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00725         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00726
00727 /// <summary>
00728 /// Create a new <see cref="LinearTrendLine"/> instance, specifying the coefficients.
00729 /// </summary>
00730 /// <param name="coefficients">The coefficients (<c>a0 ...  aN</c>).</param>
00731 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00732 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00733 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00734 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00735 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00736         public PolynomialTrendLine(double[] coefficients, double minX, double minY, double maxX,
      double maxY, IContinuousCoordinateSystem coordinateSystem)
00737         {
00738             Coefficients = coefficients;
00739             MinX = minX;
00740             MinY = minY;
00741             MaxX = maxX;
00742             MaxY = maxY;
00743             CoordinateSystem = coordinateSystem;
00744         }
00745
00746         private static double[] FitPolynomialWithFixedIntercept(double[] x, double[] y, int order,
      double fixedIntercept)
00747         {
00748             Matrix<double> v = new DenseMatrix(x.Length, order);
00749
00750             for (int i = 0; i < v.RowCount; i++)
00751             {
00752                 for (int j = 0; j < order; j++)
00753                 {
00754                     v[i, j] = Math.Pow(x[i], j + 1);
00755                 }
00756             }
00757
00758             double[] coeffs = MultipleRegression.QR(v, Vector<double>.Build.Dense(y)).ToArray();
00759             double[] tbr = new double[coeffs.Length + 1];
00760             tbr[0] = fixedIntercept;
00761             for (int i = 0; i < coeffs.Length; i++)
00762             {
00763                 tbr[i + 1] = coeffs[i];
00764             }
00765
00766             return tbr;
00767         }
00768
00769 /// <summary>
00770 /// Create a new <see cref="PolynomialTrendLine"/> instance, determining the coefficients by running a
      regression.
```

```
00771 /// </summary>
00772 /// <param name="data">The data that will be used to determine the coefficients.</param>
00773 /// <param name="order">The order of the polynomial (<c>N</c>).  This must be ≥ 2.</param>
00774 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00775 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (a0) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00776 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="order"/> is &lt;
      2.</exception>
00777         public PolynomialTrendLine(IReadOnlyList<IReadOnlyList<double» data, int order,
      IContinuousCoordinateSystem coordinateSystem,  double?  fixedIntercept = null)
00778         {
00779             if (order < 2)
00780             {
00781                 throw new ArgumentOutOfRangeException(nameof(order), order, "The polynomial order must
      be 2 or greater!");
00782             }
00783 
00784             MinX = double.MaxValue;
00785             MinY = double.MaxValue;
00786             MaxX = double.MinValue;
00787             MaxY = double.MinValue;
00788 
00789             double[] x = new double[data.Count];
00790             double[] y = new double[data.Count];
00791 
00792             double shift = 0;
00793 
00794             if (fixedIntercept != null)
00795             {
00796                 shift = fixedIntercept.Value;
00797             }
00798 
00799             for (int i = 0; i < data.Count; i++)
00800             {
00801                 MinX = Math.Min(MinX, data[i][0]);
00802                 MinY = Math.Min(MinY, data[i][1]);
00803                 MaxX = Math.Max(MaxX, data[i][0]);
00804                 MaxY = Math.Max(MaxY, data[i][1]);
00805 
00806                 x[i] = data[i][0];
00807                 y[i] = data[i][1] - shift;
00808             }
00809 
00810             if (fixedIntercept == null)
00811             {
00812                 Coefficients = MathNet.Numerics.Fit.Polynomial(x, y, order);
00813             }
00814             else
00815             {
00816                 Coefficients = FitPolynomialWithFixedIntercept(x, y, order, fixedIntercept.Value);
00817             }
00818 
00819             this.CoordinateSystem = coordinateSystem;
00820         }
00821 
00822 /// <summary>
00823 /// Create a new <see cref="PolynomialTrendLine"/> instance, determining the coefficients by running a
      regression.
00824 /// </summary>
00825 /// <param name="data">The data that will be used to determine the coefficients.</param>
00826 /// <param name="order">The order of the polynomial (<c>N</c>).  This must be ≥ 2.</param>
00827 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00828 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (a0) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00829 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="order"/> is &lt;
      2.</exception>
00830         public PolynomialTrendLine(IReadOnlyList<(double, double)> data, int order,
      IContinuousCoordinateSystem coordinateSystem,  double?  fixedIntercept = null) :  this((from el in
      data select new double[] { el.Item1, el.Item2 }).ToArray(), order, coordinateSystem, fixedIntercept) {
      }
00831 
00832 /// <inheritdoc/>
00833         public void Plot(Graphics target)
00834         {
00835             double startX = Math.Min(MinX, MaxX);
00836             double endX = Math.Max(MinX, MaxX);
00837 
00838             int count = (int)Math.Ceiling((endX - startX) / CoordinateSystem.Resolution[0]);
00839 
00840             GraphicsPath path = new GraphicsPath();
00841 
00842             for (int i = 0; i <= count; i++)
00843             {
00844                 double[] pt = new double[] { startX + (endX - startX) / count * i,
      Polynomial.Evaluate(startX + (endX - startX) / count * i, Coefficients) };
```

```
00845
00846                 if (i == 0)
00847                 {
00848                     path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00849                 }
00850                 else
00851                 {
00852                     path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00853                 }
00854             }
00855
00856             Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00857             Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00858             Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00859             Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00860
00861             double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
00862             double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00863             double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00864             double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00865
00866             target.Save();
00867             target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00868
00869             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      Tag);
00870             target.Restore();
00871         }
00872     }
00873
00874 /// <summary>
00875 /// A plot element that draws a power law trendline with equation <c>y = b * x^a</c>.
00876 /// </summary>
00877     public class PowerLawTrendLine :  IPlotElement
00878     {
00879 /// <summary>
00880 /// The slope of the trendline (a).
00881 /// </summary>
00882         public double Slope { get; set; }
00883
00884 /// <summary>
00885 /// The intercept of the trendline (b).
00886 /// </summary>
00887         public double Intercept { get; set; }
00888
00889 /// <summary>
00890 /// The minimum X value for which the trendline is plotted.
00891 /// </summary>
00892         public double MinX { get; set; }
00893
00894 /// <summary>
00895 /// The minimum Y value for which the trendline is plotted.
00896 /// </summary>
00897         public double MinY { get; set; }
00898
00899 /// <summary>
00900 /// The maximum X value for which the trendline is plotted.
00901 /// </summary>
00902         public double MaxX { get; set; }
00903
00904 /// <summary>
00905 /// The maximum Y value for which the trendline is plotted.
00906 /// </summary>
00907         public double MaxY { get; set; }
00908
00909 /// <summary>
00910 /// Presentation attributes for the trendline.
00911 /// </summary>
00912         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00913
00914 /// <summary>
00915 /// A tag to identify the trendline in the plot.
00916 /// </summary>
00917         public string Tag { get; set; }
00918
00919 /// <summary>
00920 /// The coordinate system used to transform the points from data space to plot space.
00921 /// </summary>
00922         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00923         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
```

```
00924
00925 /// <summary>
00926 /// Create a new <see cref="PowerLawTrendLine"/> instance, specifying the equation parameters.
00927 /// </summary>
00928 /// <param name="slope">The slope of the trendline (a).</param>
00929 /// <param name="intercept">The intercept of the trendline (b).</param>
00930 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00931 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00932 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00933 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00934 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00935         public PowerLawTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00936         {
00937             Slope = slope;
00938             Intercept = intercept;
00939             MinX = minX;
00940             MinY = minY;
00941             MaxX = maxX;
00942             MaxY = maxY;
00943             CoordinateSystem = coordinateSystem;
00944         }
00945
00946 /// <summary>
00947 /// Create a new <see cref="PowerLawTrendLine"/> instance, determining the equation parameters by
      running a regression.
00948 /// </summary>
00949 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00950 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00951         public PowerLawTrendLine(IReadOnlyList<IReadOnlyList<double» data, IContinuousCoordinateSystem
      coordinateSystem)
00952         {
00953             MinX = double.MaxValue;
00954             MinY = double.MaxValue;
00955             MaxX = double.MinValue;
00956             MaxY = double.MinValue;
00957
00958             double[] x = new double[data.Count];
00959             double[] y = new double[data.Count];
00960
00961             for (int i = 0; i < data.Count; i++)
00962             {
00963                 MinX = Math.Min(MinX, data[i][0]);
00964                 MinY = Math.Min(MinY, data[i][1]);
00965                 MaxX = Math.Max(MaxX, data[i][0]);
00966                 MaxY = Math.Max(MaxY, data[i][1]);
00967
00968                 x[i] = Math.Log(data[i][0]);
00969                 y[i] = Math.Log(data[i][1]);
00970             }
00971
00972             (double intercept, double slope) =
      MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00973
00974             Slope = slope;
00975             Intercept = Math.Exp(intercept);
00976
00977             this.CoordinateSystem = coordinateSystem;
00978         }
00979
00980 /// <summary>
00981 /// Create a new <see cref="PowerLawTrendLine"/> instance, determining the equation parameters by
      running a regression.
00982 /// </summary>
00983 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00984 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00985         public PowerLawTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem) :  this((from el in data select new double[] { el.Item1, el.Item2 }).ToArray(),
      coordinateSystem) { }
00986
00987 /// <inheritdoc/>
00988         public void Plot(Graphics target)
00989         {
00990             double y0 = Intercept * Math.Pow(MinX, Slope);
00991             double x0 = Math.Pow(MinY / Intercept, 1 / Slope);
00992
00993             double y1 = Intercept * Math.Pow(MaxX, Slope);
00994             double x1 = Math.Pow(MaxY / Intercept, 1 / Slope);
00995
00996             HashSet<(double, double)> points = new HashSet<(double, double)>();
00997
00998             if (y0 >= MinY && y0 <= MaxY)
00999             {
01000                 points.Add((MinX, y0));
```

```
01001             }
01002
01003             if (x0 >= MinX && x0 <= MaxX)
01004             {
01005                 points.Add((x0, MinY));
01006             }
01007
01008             if (y1 >= MinY && y1 <= MaxY)
01009             {
01010                 points.Add((MaxX, y1));
01011             }
01012
01013             if (x1 >= MinX && x1 <= MaxX)
01014             {
01015                 points.Add((x1, MaxY));
01016             }
01017
01018             if (points.Count == 2)
01019             {
01020                 double[][] uniquePoints = new double[points.Count][];
01021
01022                 int index = 0;
01023
01024                 foreach ((double x, double y) in points)
01025                 {
01026                     uniquePoints[index] = new double[] { x, y };
01027                     index++;
01028                 }
01029
01030                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
01031                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
01032
01033                 int count = (int)Math.Ceiling(Math.Max((endX - startX) /
      CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) /
      CoordinateSystem.Resolution[1]));
01034
01035                 GraphicsPath path = new GraphicsPath();
01036
01037                 for (int i = 0; i <= count; i++)
01038                 {
01039                     double[] pt = new double[] { startX + (endX - startX) / count * i, Intercept *
      Math.Pow(startX + (endX - startX) / count * i, Slope) };
01040
01041                     if (i == 0)
01042                     {
01043                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
01044                     }
01045                     else
01046                     {
01047                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
01048                     }
01049                 }
01050
01051                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
01052                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
01053                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
01054                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
01055
01056                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
01057                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
01058                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
01059                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
01060
01061                 target.Save();
01062                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
01063
01064                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, Tag);
01065                 target.Restore();
01066             }
01067
01068         }
01069     }
01070
01071 /// <summary>
01072 /// A plot element that draws a moving average trendline.
01073 /// </summary>
01074     public class MovingAverageTrendLine :  IPlotElement
01075     {
01076 /// <summary>
01077 /// The data used to compute the moving average.  These must be sorted in a sensible way.
01078 /// </summary>
```

```
01079         public IReadOnlyList<IReadOnlyList<double» Data { get; set; }
01080
01081 /// <summary>
01082 /// The weight function.  This should accept two parameters:  an <see langword="int"/> representing
      the difference in
01083 /// index between the point being weighted and the "focus point", and a <see langword="double"/>[]
      representing the
01084 /// difference in coordinates between the two points.  It should return a <see langword="double"/>
      representing the
01085 /// weight (it does not need to be normalised).
01086 /// </summary>
01087         public Func<int, double[], double> Weight { get; set; } = (int i, double[] d) => 1;
01088
01089 /// <summary>
01090 /// The number of points that are averaged to obtain the value for each point.  This must be ≥ 0.
01091 /// </summary>
01092         public int Period { get; set; }
01093
01094 /// <summary>
01095 /// Presentation attributes for the trendline.
01096 /// </summary>
01097         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
01098
01099 /// <summary>
01100 /// A tag to identify the trendline in the plot.
01101 /// </summary>
01102         public string Tag { get; set; }
01103
01104 /// <summary>
01105 /// The coordinate system used to transform the points from data space to plot space.
01106 /// </summary>
01107         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
01108         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
01109
01110 /// <summary>
01111 /// Create a new <see cref="MovingAverageTrendLine"/> instance from the specified data.
01112 /// </summary>
01113 /// <param name="data">The data used to compute the moving average.  These must be sorted in a
      sensible way.</param>
01114 /// <param name="period">The number of points that are averaged to obtain the value for each point.
      This must be ≥ 0.</param>
01115 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01116 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="period"/> is &lt;
      0.</exception>
01117         public MovingAverageTrendLine(IReadOnlyList<IReadOnlyList<double» data, int period,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
01118         {
01119             if (period < 0)
01120             {
01121                 throw new ArgumentOutOfRangeException(nameof(period), period, "The period of the
      moving average must be greater than or equal to zero!");
01122             }
01123
01124             this.Data = data;
01125             this.Period = period;
01126             CoordinateSystem = coordinateSystem;
01127         }
01128
01129 /// <summary>
01130 /// Create a new <see cref="MovingAverageTrendLine"/> instance from the specified data.
01131 /// </summary>
01132 /// <param name="data">The data used to compute the moving average.  These must be sorted in a
      sensible way.</param>
01133 /// <param name="period">The number of points that are averaged to obtain the value for each point.
      This must be ≥ 0.</param>
01134 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01135 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="period"/> is &lt;
      0.</exception>
01136         public MovingAverageTrendLine(IReadOnlyList<(double, double)> data, int period,
      IContinuousCoordinateSystem coordinateSystem) :  this((from el in data select new double[] { el.Item1,
      el.Item2 }).ToArray(), period, coordinateSystem) { }
01137
01138 /// <inheritdoc/>
01139         public void Plot(Graphics target)
01140         {
01141             GraphicsPath path = new GraphicsPath();
01142
01143             for (int i = 0; i < Data.Count; i++)
01144             {
01145                 double count = 0;
01146
01147                 double[] point = new double[Data[i].Count];
01148
```

```
01149                 for (int k = -Period; k <= Period; k++)
01150                 {
01151                     if (i + k >= 0 && i + k < Data.Count)
01152                     {
01153                         double[] diff = new double[Data[i].Count];
01154 
01155                         for (int j = 0; j < Data[i].Count; j++)
01156                         {
01157                             diff[j] = Data[i + k][j] - Data[i][j];
01158                         }
01159 
01160                         double weighting = Weight(k, diff);
01161 
01162                         count += weighting;
01163 
01164                         for (int j = 0; j < point.Length; j++)
01165                         {
01166                             point[j] += Data[i + k][j] * weighting;
01167                         }
01168                     }
01169                 }
01170 
01171                 for (int j = 0; j < point.Length; j++)
01172                 {
01173                     point[j] /= count;
01174                 }
01175 
01176                 if (i == 0)
01177                 {
01178                     path.MoveTo(CoordinateSystem.ToPlotCoordinates(point));
01179                 }
01180                 else
01181                 {
01182                     path.LineTo(CoordinateSystem.ToPlotCoordinates(point));
01183                 }
01184             }
01185 
01186             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      Tag);
01187 
01188         }
01189     }
01190 }
```

8.40 Violin.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// A plot element that draws a violin plot.
00026 /// </summary>
00027     public class Violin :  IPlotElement
00028     {
00029 /// <summary>
00030 /// The sides on which a violin can be drawn.
00031 /// </summary>
00032         public enum ViolinSide
00033         {
00034 /// <summary>
00035 /// Draw the violin only on the left side.
00036 /// </summary>
```

```
00037             Left,
00038
00039 /// <summary>
00040 /// Draw the violin only on the right side.
00041 /// </summary>
00042             Right,
00043
00044 /// <summary>
00045 /// Draw the violin on both left and right sides.
00046 /// </summary>
00047             Both
00048         }
00049
00050 /// <summary>
00051 /// The position of the origin of the violin (e.g., the 0 in data space coordinates).
00052 /// </summary>
00053         public IReadOnlyList<double> Position { get; set; }
00054
00055 /// <summary>
00056 /// The direction along which the violin is drawn, in data space coordinates.
00057 /// </summary>
00058         public IReadOnlyList<double> Direction { get; set; }
00059
00060 /// <summary>
00061 /// The width of the violin in data space coordinates.
00062 /// </summary>
00063         public double Width { get; set; } = 10;
00064
00065 /// <summary>
00066 /// Determines whether the violin is smoothed or not.
00067 /// </summary>
00068         public bool Smooth { get; set; } = true;
00069
00070 /// <summary>
00071 /// Determines on which side(s) the violin is drawn.
00072 /// </summary>
00073         public ViolinSide Sides { get; set; } = ViolinSide.Both;
00074
00075 /// <summary>
00076 /// The values whose distribution is displayed by the violin plot.
00077 /// </summary>
00078         public IReadOnlyList<double> Data { get; set; }
00079
00080 /// <summary>
00081 /// Maximum number of bins.
00082 /// </summary>
00083         public int MaxBins { get; set; } = int.MaxValue;
00084
00085 /// <summary>
00086 /// Presentation attributes for the violin plot.
00087 /// </summary>
00088         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = Colours.White };
00089
00090 /// <summary>
00091 /// The coordinate system used to transform the points from data space to plot space.
00092 /// </summary>
00093         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00094         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00095
00096 /// <summary>
00097 /// A tag to identify the violin in the plot.
00098 /// </summary>
00099         public string Tag { get; set; }
00100
00101 /// <summary>
00102 /// Create a new <see cref="Violin"/> instance.
00103 /// </summary>
00104 /// <param name="position">The position of the origin of the violin (e.g., the 0 in data space
      coordinates).</param>
00105 /// <param name="direction">The direction along which the violin is drawn, in data space
      coordinates.</param>
00106 /// <param name="data">The values whose distribution is displayed by the violin plot.</param>
00107 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00108         public Violin(IReadOnlyList<double> position, IReadOnlyList<double> direction,
      IReadOnlyList<double> data, ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00109         {
00110             this.Position = position;
00111             this.Direction = direction;
00112             this.Data = data;
00113             this.CoordinateSystem = coordinateSystem;
00114         }
00115
00116 /// <inheritdoc/>
00117         public void Plot(Graphics target)
00118         {
```

```
00119
00120             double min = double.MaxValue;
00121             double max = double.MinValue;
00122
00123             for (int i = 0; i < Data.Count; i++)
00124             {
00125                 min = Math.Min(min, Data[i]);
00126                 max = Math.Max(max, Data[i]);
00127             }
00128
00129             (double _, double _, double iqr) = Plots.Plot.IQR(Data);
00130             double h2 = 2 * iqr / Math.Pow(Data.Count, 1.0 / 3.0);
00131             int binCount = Math.Max(1, (int)Math.Ceiling((max - min) / h2));
00132
00133             binCount = Math.Min(binCount, MaxBins);
00134
00135             int[] bins = new int[binCount];
00136
00137             if (max > min)
00138             {
00139                 for (int i = 0; i < Data.Count; i++)
00140                 {
00141                     int index = (int)Math.Min(binCount - 1, Math.Floor((Data[i] - min) / (max - min) *
      binCount));
00142
00143                     bins[index]++;
00144                 }
00145             }
00146             else
00147             {
00148                 bins[0] = Data.Count;
00149             }
00150
00151             double binMax = bins.Max();
00152
00153             double[] perpDirection = new double[] { Direction[1], -Direction[0] };
00154
00155             List<Point> points = new List<Point>();
00156             List<Point> pointsOpposite = new List<Point>();
00157
00158             Point topPoint = CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * min, Position[1] + Direction[1] * min });
00159             Point bottomPoint = CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * max, Position[1] + Direction[1] * max });
00160
00161             for (int i = 0; i < bins.Length; i++)
00162             {
00163                 double binStart = min + (max - min) / binCount * i;
00164                 double binEnd = min + (max - min) / binCount * (i + 1);
00165
00166                 double x = (binStart + binEnd) * 0.5;
00167
00168                 if (i == 0)
00169                 {
00170                     if (bins.Length > 1 && bins[i + 1] > bins[i])
00171                     {
00172                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binStart, Position[1] + Direction[1] * binStart }));
00173                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binStart, Position[1] + Direction[1] * binStart }));
00174                     }
00175                     else
00176                     {
00177                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binStart + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] *
      binStart + perpDirection[1] * bins[i] / binMax * Width }));
00178                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binStart - perpDirection[0] * bins[i] / binMax * Width, Position[1] +
      Direction[1] * binStart - perpDirection[1] * bins[i] / binMax * Width }));
00179                     }
00180                 }
00181
00182                 points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * x + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] * x +
      perpDirection[1] * bins[i] / binMax * Width }));
00183                 pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * x - perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] * x -
      perpDirection[1] * bins[i] / binMax * Width }));
00184
00185                 if (i == bins.Length - 1)
00186                 {
00187                     if (bins.Length > 1 && bins[i - 1] > bins[i])
00188                     {
00189                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binEnd, Position[1] + Direction[1] * binEnd }));
00190                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binEnd, Position[1] + Direction[1] * binEnd }));
```

```
00191                     }
00192                     else
00193                     {
00194                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binEnd + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] *
      binEnd + perpDirection[1] * bins[i] / binMax * Width }));
00195                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binEnd - perpDirection[0] * bins[i] / binMax * Width, Position[1] +
      Direction[1] * binEnd - perpDirection[1] * bins[i] / binMax * Width }));
00196                     }
00197                 }
00198             }
00199
00200             pointsOpposite.Reverse();
00201
00202             GraphicsPath path = new GraphicsPath();
00203
00204             if (Sides == ViolinSide.Left || Sides == ViolinSide.Both)
00205             {
00206                 if (Smooth)
00207                 {
00208                     path.AddSmoothSpline(pointsOpposite.ToArray());
00209                 }
00210                 else
00211                 {
00212                     for (int i= 0; i < pointsOpposite.Count; i++)
00213                     {
00214                         if (i > 0)
00215                         {
00216                             path.LineTo(pointsOpposite[i]);
00217                         }
00218                         else
00219                         {
00220                             path.MoveTo(pointsOpposite[i]);
00221                         }
00222                     }
00223                 }
00224             }
00225             else
00226             {
00227                 path.MoveTo(bottomPoint).LineTo(topPoint);
00228             }
00229
00230             if (Sides == ViolinSide.Right || Sides == ViolinSide.Both)
00231             {
00232                 if (Smooth)
00233                 {
00234                     path.AddSmoothSpline(points.ToArray());
00235                 }
00236                 else
00237                 {
00238                     for (int i = 0; i < points.Count; i++)
00239                     {
00240                         path.LineTo(points[i]);
00241                     }
00242                 }
00243             }
00244             else
00245             {
00246                 path.LineTo(topPoint).LineTo(bottomPoint);
00247             }
00248
00249             path.Close();
00250
00251             string tag = Tag;
00252
00253             string strokeTag = Tag;
00254
00255             if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00256             {
00257                 strokeTag += "@stroke";
00258             }
00259
00260
00261             if (PresentationAttributes.Fill != null)
00262             {
00263                 target.FillPath(path, PresentationAttributes.Fill, tag);
00264             }
00265
00266             if (PresentationAttributes.Stroke != null)
00267             {
00268                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00269             }
00270         }
00271     }
```

```
00272 }
```

8.41 GradientBrushApplicator.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Buffers;
00020 using System.Numerics;
00021 using SixLabors.ImageSharp.Memory;
00022 using SixLabors.ImageSharp.PixelFormats;
00023 using SixLabors.ImageSharp.Drawing.Processing;
00024 using SixLabors.ImageSharp;
00025 using System.Reflection;
00026
00027 namespace VectSharp.Raster.ImageSharp
00028 {
00029     // Adapted from
      https://github.com/SixLabors/ImageSharp.Drawing/blob/master/src/ImageSharp.Drawing/Processing/GradientBrush.cs
      - Why isn't this public?  D:
00030     internal abstract class GradientBrushApplicator<TPixel> :  BrushApplicator<TPixel>
00031             where TPixel :  unmanaged, IPixel<TPixel>
00032     {
00033         private static readonly TPixel Transparent = Color.Transparent.ToPixel<TPixel>();
00034
00035         private readonly ColorStop[] colorStops;
00036
00037         private readonly GradientRepetitionMode repetitionMode;
00038
00039         private readonly MemoryAllocator allocator;
00040
00041         private readonly int scalineWidth;
00042
00043         private readonly object blenderBuffers;
00044
00045         private bool isDisposed;
00046
00047         private PixelBlender<TPixel> _blender;
00048
00049         IMemoryOwner<float> amountBuffer;
00050         IMemoryOwner<TPixel> overlayBuffer;
00051         static Type constructedLocalBlenderBuffersType;
00052
00053         static GradientBrushApplicator()
00054         {
00055             Assembly utilities = Assembly.Load("SixLabors.ImageSharp.Drawing");
00056             Type genericThreadLocalBlenderBuffersType =
      utilities.GetType("SixLabors.ImageSharp.Drawing.Utilities.ThreadLocalBlenderBuffers`1", true);
00057
00058             Type[] typeArgs = new Type[] { typeof(TPixel) };
00059             constructedLocalBlenderBuffersType =
      genericThreadLocalBlenderBuffersType.MakeGenericType(typeArgs);
00060         }
00061
00062
00063 /// <summary>
00064 /// Initializes a new instance of the <see cref="GradientBrushApplicator{TPixel}"/> class.
00065 /// </summary>
00066 /// <param name="configuration">The configuration instance to use when performing operations.</param>
00067 /// <param name="options">The graphics options.</param>
00068 /// <param name="target">The target image.</param>
00069 /// <param name="colorStops">An array of color stops sorted by their position.</param>
00070 /// <param name="repetitionMode">Defines if and how the gradient should be repeated.</param>
00071         protected GradientBrushApplicator(
00072             Configuration configuration,
00073             GraphicsOptions options,
00074             ImageFrame<TPixel> target,
00075             ColorStop[] colorStops,
```

```
00076             GradientRepetitionMode repetitionMode)
00077             :  base(configuration, options, target)
00078         {
00079             // TODO: requires colorStops to be sorted by position.
00080             // Use Array.Sort with a custom comparer.
00081             this.colorStops = colorStops;
00082             this.repetitionMode = repetitionMode;
00083             this.scalineWidth = target.Width;
00084             this.allocator = configuration.MemoryAllocator;
00085 
00086             this.blenderBuffers = Activator.CreateInstance(constructedLocalBlenderBuffersType, new
      object[] { this.allocator, this.scalineWidth, false });
00087 
00088             this._blender = (PixelBlender<TPixel>)this.GetType().GetProperty("Blender",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetProperty).GetValue(this);
00089 
00090             object data = this.blenderBuffers.GetType().GetField("data", BindingFlags.Instance |
      BindingFlags.NonPublic | BindingFlags.GetField).GetValue(this.blenderBuffers);
00091             object bufferOwner = data.GetType().GetProperty("Value", BindingFlags.Instance |
      BindingFlags.Public | BindingFlags.GetProperty).GetValue(data);
00092             amountBuffer = (IMemoryOwner<float>)bufferOwner.GetType().GetField("amountBuffer",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetField).GetValue(bufferOwner);
00093             overlayBuffer = (IMemoryOwner<TPixel>)bufferOwner.GetType().GetField("overlayBuffer",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetField).GetValue(bufferOwner);
00094         }
00095 
00096         internal TPixel this[int x, int y]
00097         {
00098             get
00099             {
00100                 float positionOnCompleteGradient = this.PositionOnGradient(x + 0.5f, y + 0.5f);
00101 
00102                 switch (this.repetitionMode)
00103                 {
00104                     case GradientRepetitionMode.None:
00105                         // do nothing.  The following could be done, but is not necessary:
00106                         // onLocalGradient = Math.Min(0, Math.Max(1, onLocalGradient));
00107                         break;
00108                     case GradientRepetitionMode.Repeat:
00109                         positionOnCompleteGradient %= 1;
00110                         break;
00111                     case GradientRepetitionMode.Reflect:
00112                         positionOnCompleteGradient %= 2;
00113                         if (positionOnCompleteGradient > 1)
00114                         {
00115                             positionOnCompleteGradient = 2 - positionOnCompleteGradient;
00116                         }
00117 
00118                         break;
00119                     case GradientRepetitionMode.DontFill:
00120                         if (positionOnCompleteGradient > 1 || positionOnCompleteGradient < 0)
00121                         {
00122                             return Transparent;
00123                         }
00124 
00125                         break;
00126                     default:
00127                         throw new ArgumentOutOfRangeException();
00128                 }
00129 
00130                 (ColorStop from, ColorStop to) = this.GetGradientSegment(positionOnCompleteGradient);
00131 
00132                 if (from.Color.Equals(to.Color))
00133                 {
00134                     return from.Color.ToPixel<TPixel>();
00135                 }
00136 
00137                 float onLocalGradient = (positionOnCompleteGradient - from.Ratio) / (to.Ratio -
      from.Ratio);
00138 
00139                 return new Color(Vector4.Lerp((Vector4)from.Color, (Vector4)to.Color,
      onLocalGradient)).ToPixel<TPixel>();
00140             }
00141         }
00142 
00143 /// <inheritdoc />
00144         public override void Apply(Span<float> scanline, int x, int y)
00145         {
00146             Span<float> amounts = amountBuffer.Memory.Span.Slice(0, scanline.Length);
00147             Span<TPixel> overlays = overlayBuffer.Memory.Span.Slice(0, scanline.Length);
00148 
00149             float blendPercentage = this.Options.BlendPercentage;
00150 
00151             // TODO: Remove bounds checks.
00152             if (blendPercentage < 1)
00153             {
00154                 for (int i = 0; i < scanline.Length; i++)
```

```
00155                 {
00156                     amounts[i] = scanline[i] * blendPercentage;
00157                     overlays[i] = this[x + i, y];
00158                 }
00159             }
00160             else
00161             {
00162                 for (int i = 0; i < scanline.Length; i++)
00163                 {
00164                     amounts[i] = scanline[i];
00165                     overlays[i] = this[x + i, y];
00166                 }
00167             }
00168 
00169             Span<TPixel> destinationRow = this.Target.PixelBuffer.DangerousGetRowSpan(y).Slice(x,
      scanline.Length);
00170 
00171             this._blender.Blend(this.Configuration, destinationRow, destinationRow, overlays,
      amounts);
00172         }
00173 
00174 /// <summary>
00175 /// Calculates the position on the gradient for a given point.
00176 /// This method is abstract as it's content depends on the shape of the gradient.
00177 /// </summary>
00178 /// <param name="x">The x-coordinate of the point.</param>
00179 /// <param name="y">The y-coordinate of the point.</param>
00180 /// <returns>
00181 /// The position the given point has on the gradient.
00182 /// The position is not bound to the [0..1] interval.
00183 /// Values outside of that interval may be treated differently,
00184 /// e.g.  for the <see cref="GradientRepetitionMode" /> enum.
00185 /// </returns>
00186         protected abstract float PositionOnGradient(float x, float y);
00187 
00188 /// <inheritdoc/>
00189         protected override void Dispose(bool disposing)
00190         {
00191             if (this.isDisposed)
00192             {
00193                 return;
00194             }
00195 
00196             base.Dispose(disposing);
00197 
00198             if (disposing)
00199             {
00200                 ((IDisposable)this.blenderBuffers).Dispose();
00201             }
00202 
00203             this.isDisposed = true;
00204         }
00205 
00206         private (ColorStop From, ColorStop To) GetGradientSegment(float positionOnCompleteGradient)
00207         {
00208             ColorStop localGradientFrom = this.colorStops[0];
00209             ColorStop localGradientTo = default;
00210 
00211             // TODO: ensure colorStops has at least 2 items (technically 1 would be okay, but that's
      no gradient)
00212             foreach (ColorStop colorStop in this.colorStops)
00213             {
00214                 localGradientTo = colorStop;
00215 
00216                 if (colorStop.Ratio > positionOnCompleteGradient)
00217                 {
00218                     // we're done here, so break it!
00219                     break;
00220                 }
00221 
00222                 localGradientFrom = localGradientTo;
00223             }
00224 
00225             return (localGradientFrom, localGradientTo);
00226         }
00227     }
00228 }
```

8.42 ImageSharpContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
```

```
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using SixLabors.ImageSharp;
00020 using SixLabors.ImageSharp.Drawing;
00021 using SixLabors.ImageSharp.Processing;
00022 using SixLabors.ImageSharp.Drawing.Processing;
00023 using System.Collections.Generic;
00024 using SixLabors.ImageSharp.Advanced;
00025 using System.IO;
00026 using VectSharp.Filters;
00027 using SixLabors.ImageSharp.Formats.Gif;
00028 using System.Threading.Tasks;
00029
00030 namespace VectSharp.Raster.ImageSharp
00031 {
00032     internal static class MatrixUtils
00033     {
00034         public static System.Numerics.Matrix3x2 ToMatrix(this double[,] matrix)
00035         {
00036             return new System.Numerics.Matrix3x2((float)matrix[0, 0], (float)matrix[1, 0],
      (float)matrix[0, 1], (float)matrix[1, 1], (float)matrix[0, 2], (float)matrix[1, 2]);
00037         }
00038
00039         public static double[] Multiply(double[,] matrix, double[] vector)
00040         {
00041             double[] tbr = new double[2];
00042
00043             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00044             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00045
00046             return tbr;
00047         }
00048
00049         public static Point Multiply(this double[,] matrix, Point vector)
00050         {
00051             return new Point(matrix[0, 0] * vector.X + matrix[0, 1] * vector.Y + matrix[0, 2],
      matrix[1, 0] * vector.X + matrix[1, 1] * vector.Y + matrix[1, 2]);
00052         }
00053
00054         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00055         {
00056             double[,] tbr = new double[3, 3];
00057
00058             for (int i = 0; i < 3; i++)
00059             {
00060                 for (int j = 0; j < 3; j++)
00061                 {
00062                     for (int k = 0; k < 3; k++)
00063                     {
00064                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00065                     }
00066                 }
00067             }
00068
00069             return tbr;
00070         }
00071
00072         public static double[,] Rotate(double[,] matrix, double angle)
00073         {
00074             double[,] rotationMatrix = new double[3, 3];
00075             rotationMatrix[0, 0] = Math.Cos(angle);
00076             rotationMatrix[0, 1] = -Math.Sin(angle);
00077             rotationMatrix[1, 0] = Math.Sin(angle);
00078             rotationMatrix[1, 1] = Math.Cos(angle);
00079             rotationMatrix[2, 2] = 1;
00080
00081             return Multiply(matrix, rotationMatrix);
00082         }
00083
00084         public static double[,] Translate(double[,] matrix, double x, double y)
00085         {
00086             double[,] translationMatrix = new double[3, 3];
00087             translationMatrix[0, 0] = 1;
00088             translationMatrix[0, 2] = x;
```

```
00089             translationMatrix[1, 1] = 1;
00090             translationMatrix[1, 2] = y;
00091             translationMatrix[2, 2] = 1;
00092 
00093             return Multiply(matrix, translationMatrix);
00094         }
00095 
00096         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00097         {
00098             double[,] scaleMatrix = new double[3, 3];
00099             scaleMatrix[0, 0] = scaleX;
00100             scaleMatrix[1, 1] = scaleY;
00101             scaleMatrix[2, 2] = 1;
00102 
00103             return Multiply(matrix, scaleMatrix);
00104         }
00105 
00106         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00107         public static double[,] Invert(double[,] m)
00108         {
00109             double[,] tbr = new double[3, 3];
00110 
00111             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00112             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00113             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00114             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00115             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00116             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00117             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00118             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00119             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00120 
00121             return tbr;
00122         }
00123 
00124         public static double Determinant(double[,] matrix)
00125         {
00126             return (matrix[0, 0] * matrix[1, 1] - matrix[1, 0] * matrix[0, 1]) * matrix[2, 2] -
      (matrix[0, 0] * matrix[1, 2] - matrix[1, 0] * matrix[0, 2]) * matrix[2, 1] + (matrix[0, 1] * matrix[1,
      2] - matrix[1, 1] * matrix[0, 2]) * matrix[2, 0];
00127         }
00128     }
00129 
00130     internal class ImageSharpContext :  IGraphicsContext
00131     {
00132         public Image<SixLabors.ImageSharp.PixelFormats.Rgba32> Image { get; }
00133         private Image Buffer { get; set; }
00134         private Image ClipBuffer { get; }
00135 
00136         public double Width { get; private set; }
00137 
00138         public double Height { get; private set; }
00139 
00140         public int IntWidth { get; private set; }
00141         public int IntHeight { get; private set; }
00142 
00143         public Font Font { get; set; }
00144         public TextBaselines TextBaseline { get; set; }
00145 
00146         public Brush FillStyle { get; set; }
00147 
00148         public Brush StrokeStyle { get; set; }
00149 
00150         public double LineWidth { get; set; }
00151         public LineCaps LineCap { get; set; }
00152         public LineJoins LineJoin { get; set; }
00153 
00154         public LineDash Dash { get; set; }
00155         public string Tag { get; set; }
```

```
00156
00157         private double[,] _transform;
00158         private readonly Stack<double[,]> states;
00159
00160         private readonly Stack<Image> clips;
00161         private Image CurrentClip;
00162
00163         private List<IDisposable> disposables;
00164
00165         PathBuilder currentPath;
00166         bool figureInitialised;
00167         Point currentPoint;
00168         double scaleFactor;
00169
00170         public ImageSharpContext(double width, double height, double scaleFactor, Colour
      backgroundColour)
00171         {
00172             currentPath = new PathBuilder();
00173             figureInitialised = false;
00174             currentPoint = new Point();
00175
00176             IntWidth = (int)(width * scaleFactor);
00177             IntHeight = (int)(height * scaleFactor);
00178
00179             this.scaleFactor = Math.Sqrt(IntWidth / width * IntHeight / height);
00180
00181             _transform = new double[3, 3];
00182
00183             _transform[0, 0] = scaleFactor;
00184             _transform[1, 1] = scaleFactor;
00185             _transform[2, 2] = 1;
00186
00187             states = new Stack<double[,]>();
00188
00189             Width = width;
00190             Height = height;
00191
00192             this.Image = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00193             this.Image.Mutate(x => x.Fill(backgroundColour.ToImageSharpColor()));
00194
00195             this.Buffer = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00196             this.Buffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00197
00198             this.ClipBuffer = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth,
      IntHeight);
00199             this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00200
00201             this.clips = new Stack<Image>();
00202             this.CurrentClip = null;
00203
00204             disposables = new List<IDisposable>();
00205         }
00206
00207         public void DisposeAllExceptImage()
00208         {
00209             for (int i = 0; i < disposables.Count; i++)
00210             {
00211                 disposables[i].Dispose();
00212                 this.CurrentClip?.Dispose();
00213                 this.Buffer.Dispose();
00214                 this.ClipBuffer.Dispose();
00215
00216             }
00217         }
00218
00219         public void Close()
00220         {
00221             this.currentPath.CloseFigure();
00222             this.figureInitialised = false;
00223         }
00224
00225         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00226         {
00227             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
00228
00229             if (figureInitialised)
00230             {
00231                 Point p1 = new Point(p1X, p1Y);
00232                 Point p2 = new Point(p2X, p2Y);
00233                 Point p3 = new Point(p3X, p3Y);
00234                 currentPath.AddCubicBezier(currentPoint.ToPointF(1), p1.ToPointF(1), p2.ToPointF(1),
      p3.ToPointF(1));
00235                 currentPoint = p3;
00236             }
00237             else
00238             {
```

```
00239                 currentPath.StartFigure();
00240                 currentPoint = new Point(p3X, p3Y);
00241                 figureInitialised = true;
00242             }
00243         }
00244 
00245         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00246         {
00247             Image sourceImage;
00248 
00249             unsafe
00250             {
00251                 if (image.HasAlpha)
00252                 {
00253                     ReadOnlySpan<byte> data = new ReadOnlySpan<byte>((void*)image.ImageDataAddress,
      image.Width * image.Height * 4);
00254                     sourceImage =
      SixLabors.ImageSharp.Image.LoadPixelData<SixLabors.ImageSharp.PixelFormats.Rgba32>(data, image.Width,
      image.Height);
00255                 }
00256                 else
00257                 {
00258                     ReadOnlySpan<byte> data = new ReadOnlySpan<byte>((void*)image.ImageDataAddress,
      image.Width * image.Height * 3);
00259                     sourceImage =
      SixLabors.ImageSharp.Image.LoadPixelData<SixLabors.ImageSharp.PixelFormats.Rgb24>(data, image.Width,
      image.Height);
00260                 }
00261             }
00262 
00263             DrawImage(sourceX, sourceY, sourceWidth, sourceHeight, destinationX, destinationY,
      destinationWidth, destinationHeight, image.Interpolate, sourceImage);
00264         }
00265 
00266         internal void DrawImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, bool
      interpolate, Image sourceImage)
00267         {
00268             sourceImage.Mutate(x => x.Crop(new SixLabors.ImageSharp.Rectangle(sourceX, sourceY,
      sourceWidth, sourceHeight)));
00269 
00270             Point targetPoint = _transform.Multiply(new Point(destinationX, destinationY));
00271             Point targetPointX = _transform.Multiply(new Point(destinationX + destinationWidth,
      destinationY));
00272             Point targetPointY = _transform.Multiply(new Point(destinationX, destinationY +
      destinationHeight));
00273             Point targetPointXY = _transform.Multiply(new Point(destinationX + destinationWidth,
      destinationY + destinationHeight));
00274 
00275             double minX = Math.Min(Math.Min(targetPoint.X, targetPointX.X), Math.Min(targetPointY.X,
      targetPointXY.X));
00276             double minY = Math.Min(Math.Min(targetPoint.Y, targetPointX.Y), Math.Min(targetPointY.Y,
      targetPointXY.Y));
00277 
00278             double maxX = Math.Max(Math.Max(targetPoint.X, targetPointX.X), Math.Max(targetPointY.X,
      targetPointXY.X));
00279             double maxY = Math.Max(Math.Max(targetPoint.Y, targetPointX.Y), Math.Max(targetPointY.Y,
      targetPointXY.Y));
00280 
00281             double[,] currTransform = MatrixUtils.Multiply(MatrixUtils.Translate(_transform,
      destinationX, destinationY), MatrixUtils.Scale(MatrixUtils.Identity, destinationWidth /
      sourceImage.Width, destinationHeight / sourceImage.Height));
00282 
00283             Point origin = currTransform.Multiply(new Point(0, 0));
00284 
00285             double[,] translation = MatrixUtils.Translate(MatrixUtils.Identity, -minX, -minY);
00286 
00287             double[,] centeredTransform = MatrixUtils.Multiply(translation, currTransform);
00288 
00289             SixLabors.ImageSharp.Processing.Processors.Transforms.IResampler resampler;
00290 
00291             if (interpolate)
00292             {
00293                 resampler = KnownResamplers.Bicubic;
00294             }
00295             else
00296             {
00297                 resampler = KnownResamplers.NearestNeighbor;
00298             }
00299 
00300             sourceImage.Mutate(x => x.Transform(new SixLabors.ImageSharp.Rectangle(sourceX, sourceY,
      sourceWidth, sourceHeight), centeredTransform.ToMatrix(), new
      SixLabors.ImageSharp.Size((int)Math.Round(maxX - minX), (int)Math.Round(maxY - minY)), resampler));
00301 
00302             if (this.CurrentClip == null)
```

```
00303             {
00304                 this.Image.Mutate(x => x.CheckedDrawImage(sourceImage, new
      SixLabors.ImageSharp.Point((int)Math.Round(minX), (int)Math.Round(minY)), 1));
00305             }
00306             else
00307             {
00308                 this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00309                 this.ClipBuffer.Mutate(x => x.CheckedDrawImage(sourceImage, new
      SixLabors.ImageSharp.Point((int)Math.Round(minX), (int)Math.Round(minY)), 1));
00310
00311                 GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00312
00313                 this.Buffer.Dispose();
00314
00315                 this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer, opt));
00316
00317                 this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
00318             }
00319         }
00320
00321         public void Fill(FillRule fillRule)
00322         {
00323             IPath path = this.currentPath.Build();
00324
00325             ShapeOptions shapeOptions = new ShapeOptions();
00326
00327             switch (fillRule)
00328             {
00329                 case FillRule.NonZeroWinding:
00330                     shapeOptions.IntersectionRule = IntersectionRule.NonZero;
00331                     break;
00332                 case FillRule.EvenOdd:
00333                     shapeOptions.IntersectionRule = IntersectionRule.EvenOdd;
00334                     break;
00335
00336             }
00337
00338             if (this.CurrentClip == null)
00339             {
00340                 this.Image.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix(), ShapeOptions = shapeOptions }, this.FillStyle.ToImageSharpBrush(_transform),
      path));
00341             }
00342             else
00343             {
00344                 this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00345                 this.ClipBuffer.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix(), ShapeOptions = shapeOptions }, this.FillStyle.ToImageSharpBrush(_transform),
      path));
00346
00347                 GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00348
00349                 this.Buffer.Dispose();
00350
00351                 this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer, opt));
00352
00353                 this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
00354             }
00355
00356             this.currentPath = new PathBuilder();
00357             this.figureInitialised = false;
00358         }
00359
00360         public void FillText(string text, double x, double y)
00361         {
00362             PathText(text, x, y);
00363             Fill(FillRule.NonZeroWinding);
00364         }
00365
00366         private void PathText(string text, double x, double y)
00367         {
00368             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00369
00370             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00371
00372             for (int j = 0; j < textPath.Segments.Count; j++)
00373             {
00374                 switch (textPath.Segments[j].Type)
00375                 {
00376                     case VectSharp.SegmentType.Move:
00377                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00378                         break;
00379                     case VectSharp.SegmentType.Line:
00380                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00381                         break;
```

```
00382                     case VectSharp.SegmentType.CubicBezier:
00383                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00384                         break;
00385                     case VectSharp.SegmentType.Close:
00386                         this.Close();
00387                         break;
00388                 }
00389             }
00390         }
00391 
00392         public void LineTo(double x, double y)
00393         {
00394             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00395 
00396             if (figureInitialised)
00397             {
00398                 Point newPoint = new Point(x, y);
00399                 currentPath.AddLine(currentPoint.ToPointF(1), newPoint.ToPointF(1));
00400                 currentPoint = newPoint;
00401             }
00402             else
00403             {
00404                 currentPath.StartFigure();
00405                 currentPoint = new Point(x, y);
00406                 figureInitialised = true;
00407             }
00408         }
00409 
00410         public void MoveTo(double x, double y)
00411         {
00412             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00413 
00414             currentPath.StartFigure();
00415             currentPoint = new Point(x, y);
00416             figureInitialised = true;
00417         }
00418 
00419         public void Rectangle(double x0, double y0, double width, double height)
00420         {
00421             MoveTo(x0, y0);
00422             LineTo(x0 + width, y0);
00423             LineTo(x0 + width, y0 + height);
00424             LineTo(x0, y0 + height);
00425             Close();
00426         }
00427 
00428         public void Restore()
00429         {
00430             _transform = states.Pop();
00431 
00432             Image newClip = clips.Pop();
00433             if (CurrentClip != newClip)
00434             {
00435                 CurrentClip.Dispose();
00436             }
00437 
00438             CurrentClip = newClip;
00439         }
00440 
00441         public void Rotate(double angle)
00442         {
00443             Utils.CoerceNaNAndInfinityToZero(ref angle);
00444 
00445             _transform = MatrixUtils.Rotate(_transform, angle);
00446 
00447             currentPath = new PathBuilder();
00448             figureInitialised = false;
00449         }
00450 
00451         public void Save()
00452         {
00453             states.Push((double[,])_transform.Clone());
00454             clips.Push(CurrentClip);
00455         }
00456 
00457         public void Scale(double x, double y)
00458         {
00459             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00460 
00461             _transform = MatrixUtils.Scale(_transform, x, y);
00462 
00463             currentPath = new PathBuilder();
00464             figureInitialised = false;
00465         }
00466 
```

```
00467         public void SetClippingPath()
00468         {
00469             IPath path = this.currentPath.Build();
00470 
00471             Image newClip;
00472 
00473             DrawingOptions opt = new DrawingOptions() { Transform = _transform.ToMatrix() };
00474 
00475             if (this.CurrentClip == null)
00476             {
00477                 newClip = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00478                 newClip.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00479                 newClip.Mutate(x => x.Fill(opt, Color.FromRgb(0, 0, 0), path));
00480             }
00481             else
00482             {
00483                 this.Buffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00484                 this.Buffer.Mutate(x => x.Fill(opt, Color.FromRgb(0, 0, 0), path));
00485 
00486                 newClip = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.Buffer, new
      GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn }));
00487 
00488                 disposables.Add(this.CurrentClip);
00489             }
00490 
00491             this.CurrentClip = newClip;
00492 
00493             this.currentPath = new PathBuilder();
00494             this.figureInitialised = false;
00495         }
00496 
00497         public void SetFillStyle((int r, int g, int b, double a) style)
00498         {
00499             this.FillStyle = Colour.FromRgba(style);
00500         }
00501 
00502         public void SetFillStyle(Brush style)
00503         {
00504             this.FillStyle = style;
00505         }
00506 
00507         public void SetLineDash(LineDash dash)
00508         {
00509             this.Dash = dash;
00510         }
00511 
00512         public void SetStrokeStyle((int r, int g, int b, double a) style)
00513         {
00514             this.StrokeStyle = Colour.FromRgba(style);
00515         }
00516 
00517         public void SetStrokeStyle(Brush style)
00518         {
00519             this.StrokeStyle = style;
00520         }
00521 
00522         public void Stroke()
00523         {
00524             if (this.LineWidth > 0)
00525             {
00526                 if (this.CurrentClip == null)
00527                 {
00528                     this.Image.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix() }, this.StrokeStyle.ToImageSharpBrush(_transform),
      this.currentPath.Build().GenerateOutline((float)(this.LineWidth),
      this.Dash.ToImageSharpDash(this.LineWidth), false, this.LineJoin.ToImageSharpJoint(),
      this.LineCap.ToImageSharpCap())));
00529                 }
00530                 else
00531                 {
00532                     this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00533                     this.ClipBuffer.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix() }, this.StrokeStyle.ToImageSharpBrush(_transform),
      this.currentPath.Build().GenerateOutline((float)(this.LineWidth),
      this.Dash.ToImageSharpDash(this.LineWidth), false, this.LineJoin.ToImageSharpJoint(),
      this.LineCap.ToImageSharpCap())));
00534 
00535                     GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00536 
00537                     this.Buffer.Dispose();
00538 
00539                     this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer,
      opt));
00540 
00541                     this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
```

```
00542                 }
00543             }
00544
00545             this.currentPath = new PathBuilder();
00546             this.figureInitialised = false;
00547         }
00548
00549         public void StrokeText(string text, double x, double y)
00550         {
00551             PathText(text, x, y);
00552             Stroke();
00553         }
00554
00555         public void Transform(double a, double b, double c, double d, double e, double f)
00556         {
00557             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
00558
00559             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00560             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
00561
00562             currentPath = new PathBuilder();
00563             figureInitialised = false;
00564         }
00565
00566         public void Translate(double x, double y)
00567         {
00568             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00569
00570             _transform = MatrixUtils.Translate(_transform, x, y);
00571
00572             currentPath = new PathBuilder();
00573             figureInitialised = false;
00574         }
00575
00576         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00577         {
00578             double scale = this.scaleFactor * Math.Sqrt(MatrixUtils.Determinant(_transform));
00579
00580             Rectangle bounds = graphics.GetBounds();
00581
00582             bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00583
00584             if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00585             {
00586                 bool rasterisationNeeded = true;
00587
00588                 if (filter is GaussianBlurFilter gauss)
00589                 {
00590                     rasterisationNeeded = false;
00591
00592                     Page pag = new Page(1, 1);
00593                     pag.Graphics.DrawGraphics(0, 0, graphics);
00594                     pag.Crop(bounds.Location, bounds.Size);
00595
00596                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00597                     img.Mutate(x => x.GaussianBlur((float)(gauss.StandardDeviation * scale)));
00598
00599                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00600
00601                     img.Dispose();
00602                 }
00603                 else if (filter is ColourMatrixFilter cmf)
00604                 {
00605                     rasterisationNeeded = false;
00606
00607                     Page pag = new Page(1, 1);
00608                     pag.Graphics.DrawGraphics(0, 0, graphics);
00609                     pag.Crop(bounds.Location, bounds.Size);
00610
00611                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00612
00613                     img.Mutate(x => x.Filter(new ColorMatrix((float)cmf.ColourMatrix.R1,
      (float)cmf.ColourMatrix.G1, (float)cmf.ColourMatrix.B1, (float)cmf.ColourMatrix.A1,
      (float)cmf.ColourMatrix.R2, (float)cmf.ColourMatrix.G2, (float)cmf.ColourMatrix.B2,
      (float)cmf.ColourMatrix.A2, (float)cmf.ColourMatrix.R3, (float)cmf.ColourMatrix.G3,
      (float)cmf.ColourMatrix.B3, (float)cmf.ColourMatrix.A3, (float)cmf.ColourMatrix.R4,
      (float)cmf.ColourMatrix.G4, (float)cmf.ColourMatrix.B4, (float)cmf.ColourMatrix.A4,
      (float)cmf.ColourMatrix.R5, (float)cmf.ColourMatrix.G5, (float)cmf.ColourMatrix.B5,
      (float)cmf.ColourMatrix.A5)));
00614
00615                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
```

```
00616
00617                     img.Dispose();
00618                 }
00619                 else if (filter is BoxBlurFilter bbf)
00620                 {
00621                     rasterisationNeeded = false;
00622
00623                     Page pag = new Page(1, 1);
00624                     pag.Graphics.DrawGraphics(0, 0, graphics);
00625                     pag.Crop(bounds.Location, bounds.Size);
00626
00627                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00628
00629                     img.Mutate(x => x.BoxBlur((int)Math.Round(bbf.BoxRadius * scale)));
00630
00631                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00632
00633                     img.Dispose();
00634                 }
00635                 else if (filter is CompositeLocationInvariantFilter comp)
00636                 {
00637                     bool allSupported = true;
00638
00639                     foreach (IFilter filter2 in comp.Filters)
00640                     {
00641                         if (!(filter2 is GaussianBlurFilter) && !(filter2 is ColourMatrixFilter) &&
      !(filter2 is BoxBlurFilter))
00642                         {
00643                             allSupported = false;
00644                             break;
00645                         }
00646                     }
00647
00648                     if (allSupported)
00649                     {
00650                         rasterisationNeeded = false;
00651
00652                         Page pag = new Page(1, 1);
00653                         pag.Graphics.DrawGraphics(0, 0, graphics);
00654                         pag.Crop(bounds.Location, bounds.Size);
00655
00656                         Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00657
00658                         foreach (IFilter filter2 in comp.Filters)
00659                         {
00660                             if (filter2 is GaussianBlurFilter gauss2)
00661                             {
00662                                 img.Mutate(x => x.GaussianBlur((float)(gauss2.StandardDeviation *
      scale)));
00663                             }
00664                             else if (filter2 is ColourMatrixFilter cmf2)
00665                             {
00666                                 img.Mutate(x => x.Filter(new ColorMatrix((float)cmf2.ColourMatrix.R1,
      (float)cmf2.ColourMatrix.G1, (float)cmf2.ColourMatrix.B1, (float)cmf2.ColourMatrix.A1,
      (float)cmf2.ColourMatrix.R2, (float)cmf2.ColourMatrix.G2, (float)cmf2.ColourMatrix.B2,
      (float)cmf2.ColourMatrix.A2, (float)cmf2.ColourMatrix.R3, (float)cmf2.ColourMatrix.G3,
      (float)cmf2.ColourMatrix.B3, (float)cmf2.ColourMatrix.A3, (float)cmf2.ColourMatrix.R4,
      (float)cmf2.ColourMatrix.G4, (float)cmf2.ColourMatrix.B4, (float)cmf2.ColourMatrix.A4,
      (float)cmf2.ColourMatrix.R5, (float)cmf2.ColourMatrix.G5, (float)cmf2.ColourMatrix.B5,
      (float)cmf2.ColourMatrix.A5)));
00667                             }
00668                             else if (filter2 is BoxBlurFilter bbf2)
00669                             {
00670                                 img.Mutate(x => x.BoxBlur((int)Math.Round(bbf2.BoxRadius * scale)));
00671                             }
00672                         }
00673
00674
00675                         DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00676
00677                         img.Dispose();
00678                     }
00679                 }
00680
00681                 if (rasterisationNeeded)
00682                 {
00683                     RasterImage rasterised = ImageSharpContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
00684                     RasterImage filtered = null;
00685
00686                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00687                     {
00688                         filterWithRastParam.RasteriseParameter(ImageSharpContextInterpreter.Rasterise,
```

```
scale);
00689                     }
00690
00691                     if (filter is ILocationInvariantFilter locInvFilter)
00692                     {
00693                         filtered = locInvFilter.Filter(rasterised, scale);
00694                     }
00695                     else if (filter is IFilterWithLocation filterWithLoc)
00696                     {
00697                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00698                     }
00699
00700                     if (filtered != null)
00701                     {
00702                         rasterised.Dispose();
00703
00704                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
    bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00705                     }
00706                 }
00707             }
00708         }
00709     }
00710
00711     internal static class Utils
00712     {
00713         public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
    image, SixLabors.ImageSharp.Point point, float opacity)
00714         {
00715             int x0A = 0;
00716             int y0A = 0;
00717             SixLabors.ImageSharp.Size size = x.GetCurrentSize();
00718             int x1A = size.Width;
00719             int y1A = size.Height;
00720
00721             int x0B = point.X;
00722             int y0B = point.Y;
00723             int x1B = point.X + image.Width;
00724             int y1B = point.Y + image.Height;
00725
00726             if (!(x0A >= x1B || x1A <= x0B || y0A >= y1B || y1A <= y0B))
00727             {
00728                 return x.DrawImage(image, point, opacity);
00729             }
00730             else
00731             {
00732                 return x;
00733             }
00734         }
00735
00736         public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
    image, float opacity)
00737         {
00738             return x.DrawImage(image, opacity);
00739         }
00740
00741         public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
    image, GraphicsOptions options)
00742         {
00743             return x.DrawImage(image, options);
00744         }
00745
00746         public static PointF ToPointF(this Point pt, double scaleFactor)
00747         {
00748             return new PointF((float)(pt.X * scaleFactor), (float)(pt.Y * scaleFactor));
00749         }
00750
00751         public static void CoerceNaNAndInfinityToZero(ref double val)
00752         {
00753             if (double.IsNaN(val) || double.IsInfinity(val) || val == double.MinValue || val ==
    double.MaxValue)
00754             {
00755                 val = 0;
00756             }
00757         }
00758
00759         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2)
00760         {
00761             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
    double.MaxValue)
00762             {
00763                 val1 = 0;
00764             }
00765
00766             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
    double.MaxValue)
00767             {
```

```
00768                 val2 = 0;
00769             }
00770         }
00771 
00772         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4)
00773         {
00774             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00775             {
00776                 val1 = 0;
00777             }
00778 
00779             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00780             {
00781                 val2 = 0;
00782             }
00783 
00784             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00785             {
00786                 val3 = 0;
00787             }
00788 
00789             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00790             {
00791                 val4 = 0;
00792             }
00793         }
00794 
00795         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4, ref double val5, ref double val6)
00796         {
00797             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00798             {
00799                 val1 = 0;
00800             }
00801 
00802             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00803             {
00804                 val2 = 0;
00805             }
00806 
00807             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00808             {
00809                 val3 = 0;
00810             }
00811 
00812             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00813             {
00814                 val4 = 0;
00815             }
00816 
00817             if (double.IsNaN(val5) || double.IsInfinity(val5) || val5 == double.MinValue || val5 ==
      double.MaxValue)
00818             {
00819                 val5 = 0;
00820             }
00821 
00822             if (double.IsNaN(val6) || double.IsInfinity(val6) || val6 == double.MinValue || val6 ==
      double.MaxValue)
00823             {
00824                 val6 = 0;
00825             }
00826         }
00827 
00828         public static Color ToImageSharpColor(this Colour colour)
00829         {
00830             return Color.FromRgba((byte)(colour.R * 255), (byte)(colour.G * 255), (byte)(colour.B *
      255), (byte)(colour.A * 255));
00831         }
00832 
00833         public static SixLabors.ImageSharp.Drawing.Processing.Brush ToImageSharpBrush(this Brush
      brush, double[,] transform)
00834         {
00835             if (brush is SolidColourBrush solid)
00836             {
00837                 return new
      SixLabors.ImageSharp.Drawing.Processing.SolidBrush(solid.Colour.ToImageSharpColor());
00838             }
00839             else if (brush is LinearGradientBrush linear)
```

```
00840             {
00841                 ColorStop[] colorStops = new ColorStop[linear.GradientStops.Count];
00842 
00843                 for (int i = 0; i < linear.GradientStops.Count; i++)
00844                 {
00845                     colorStops[i] = new ColorStop((float)linear.GradientStops[i].Offset,
      linear.GradientStops[i].Colour.ToImageSharpColor());
00846                 }
00847 
00848                 return new
      SixLabors.ImageSharp.Drawing.Processing.LinearGradientBrush(transform.Multiply(linear.StartPoint).ToPointF(1),
      transform.Multiply(linear.EndPoint).ToPointF(1), GradientRepetitionMode.None, colorStops);
00849             }
00850             else if (brush is RadialGradientBrush radial)
00851             {
00852                 ColorStop[] colorStops = new ColorStop[radial.GradientStops.Count];
00853 
00854                 for (int i = 0; i < radial.GradientStops.Count; i++)
00855                 {
00856                     colorStops[i] = new ColorStop((float)radial.GradientStops[i].Offset,
      radial.GradientStops[i].Colour.ToImageSharpColor());
00857                 }
00858 
00859                 return new RadialGradientBrushSVGStyle(radial.Centre, radial.FocalPoint,
      radial.Radius, transform, GradientRepetitionMode.None, colorStops);
00860             }
00861             else
00862             {
00863                 throw new NotImplementedException();
00864             }
00865         }
00866 
00867         public static float[] ToImageSharpDash(this LineDash dash, double lineThickness)
00868         {
00869             if (dash.UnitsOn > 0 || dash.UnitsOff > 0)
00870             {
00871                 return new float[]
00872                 {
00873                    (float)(dash.UnitsOn / lineThickness),
00874                     (float)(dash.UnitsOff / lineThickness)
00875                 };
00876             }
00877             else
00878             {
00879                 return new float[] { };
00880             }
00881         }
00882 
00883         public static JointStyle ToImageSharpJoint(this LineJoins join)
00884         {
00885             switch (join)
00886             {
00887                 case LineJoins.Round:
00888                     return JointStyle.Round;
00889                 case LineJoins.Miter:
00890                     return JointStyle.Miter;
00891                 case LineJoins.Bevel:
00892                     return JointStyle.Square;
00893                 default:
00894                     return JointStyle.Miter;
00895             }
00896         }
00897 
00898         public static EndCapStyle ToImageSharpCap(this LineCaps cap)
00899         {
00900             switch (cap)
00901             {
00902                 case LineCaps.Square:
00903                     return EndCapStyle.Square;
00904                 case LineCaps.Round:
00905                     return EndCapStyle.Round;
00906                 case LineCaps.Butt:
00907                     return EndCapStyle.Butt;
00908                 default:
00909                     throw new NotImplementedException();
00910             }
00911         }
00912 
00913         internal class RadialGradientBrushSVGStyle :
      SixLabors.ImageSharp.Drawing.Processing.GradientBrush
00914         {
00915             private readonly Point Centre;
00916             private readonly Point FocalPoint;
00917             private readonly double Radius;
00918             private readonly double[,] InverseTransform;
00919 
00920             public RadialGradientBrushSVGStyle(Point centre, Point focalPoint, double radius,
```

```
      double[,] transform, GradientRepetitionMode repetitionMode, params ColorStop[] colorStops) :
      base(repetitionMode, colorStops)
00921             {
00922                 Centre = centre;
00923                 FocalPoint = focalPoint;
00924                 Radius = radius;
00925                 InverseTransform = MatrixUtils.Invert(transform);
00926             }
00927 
00928             public override BrushApplicator<TPixel> CreateApplicator<TPixel>(Configuration
      configuration, GraphicsOptions options, ImageFrame<TPixel> source, RectangleF region)
00929             {
00930                 return new RadialGradientBrushPDFStyleApplicator<TPixel>(configuration, options,
      source, Centre, FocalPoint, Radius, InverseTransform, ColorStops, RepetitionMode);
00931             }
00932 
00933             private class RadialGradientBrushPDFStyleApplicator<TPixel> :
      GradientBrushApplicator<TPixel> where TPixel :  unmanaged,
      SixLabors.ImageSharp.PixelFormats.IPixel<TPixel>
00934             {
00935                 private readonly Point Centre;
00936                 private readonly Point FocalPoint;
00937                 private readonly double Radius;
00938                 private readonly double[,] InverseTransform;
00939 
00940                 private readonly double a;
00941 
00942                 public RadialGradientBrushPDFStyleApplicator(
00943                     Configuration configuration,
00944                     GraphicsOptions options,
00945                     ImageFrame<TPixel> target,
00946                     Point centre, Point focalPoint, double radius, double[,] inverseTransform,
00947                     ColorStop[] colorStops,
00948                     GradientRepetitionMode repetitionMode)
00949                 :  base(configuration, options, target, colorStops, repetitionMode)
00950                 {
00951                     this.Centre = centre;
00952                     this.FocalPoint = focalPoint;
00953                     this.Radius = radius;
00954                     this.InverseTransform = inverseTransform;
00955 
00956                     a = (centre.X - focalPoint.X) * (centre.X - focalPoint.X) + (centre.Y -
      focalPoint.Y) * (centre.Y - focalPoint.Y) - radius * radius;
00957                 }
00958 
00959                 Random rnd = new Random();
00960 
00961                 protected override float PositionOnGradient(float x, float y)
00962                 {
00963                     Point realPoint = InverseTransform.Multiply(new Point(x, y));
00964 
00965                     double c = (realPoint.X - FocalPoint.X) * (realPoint.X - FocalPoint.X) +
      (realPoint.Y - FocalPoint.Y) * (realPoint.Y - FocalPoint.Y);
00966 
00967                     double halfB = -((realPoint.X - FocalPoint.X) * (Centre.X - FocalPoint.X) +
      (realPoint.Y - FocalPoint.Y) * (Centre.Y - FocalPoint.Y));
00968 
00969                     double sqrt = Math.Sqrt(halfB * halfB - a * c);
00970 
00971                     double tbr1 = (-halfB + sqrt) / a;
00972                     double tbr2 = (-halfB - sqrt) / a;
00973 
00974                     if (tbr1 >= 0 && tbr2 < 0)
00975                     {
00976                         return (float)tbr1;
00977                     }
00978                     else if (tbr1 < 0 && tbr2 >= 0)
00979                     {
00980                         return (float)tbr2;
00981                     }
00982                     else if (tbr1 < 0 && tbr2 < 0)
00983                     {
00984                         return 0;
00985                     }
00986                     else
00987                     {
00988                         return (float)Math.Min(tbr1, tbr2);
00989                     }
00990                 }
00991 
00992 /// <inheritdoc/>
00993                 public override void Apply(Span<float> scanline, int x, int y)
00994                 {
00995                     base.Apply(scanline, x, y);
00996                 }
00997             }
00998         }
```

```
00999     }
01000
01001 /// <summary>
01002 /// Enumeration containing the supported output formats.
01003 /// </summary>
01004     public enum OutputFormats
01005     {
01006 /// <summary>
01007 /// Windows bitmap format
01008 /// </summary>
01009         BMP,
01010
01011 /// <summary>
01012 /// Graphics interchange format
01013 /// </summary>
01014         GIF,
01015
01016 /// <summary>
01017 /// Joint photographic experts group format
01018 /// </summary>
01019         JPEG,
01020
01021
01022 /// <summary>
01023 /// Portable bitmap format
01024 /// </summary>
01025         PBM,
01026
01027 /// <summary>
01028 /// Portable network graphics format
01029 /// </summary>
01030         PNG,
01031
01032 /// <summary>
01033 /// Truevision graphics adapter format
01034 /// </summary>
01035         TGA,
01036
01037 /// <summary>
01038 /// Tag image file format
01039 /// </summary>
01040         TIFF,
01041
01042 /// <summary>
01043 /// WebP format
01044 /// </summary>
01045         WebP
01046     }
01047
01048 /// <summary>
01049 /// Contains methods to render a <see cref="Page"/> to an <see cref="Image"/>.
01050 /// </summary>
01051     public static class ImageSharpContextInterpreter
01052     {
01053 /// <summary>
01054 /// Render the page to an <see cref="Image"/> object.
01055 /// </summary>
01056 /// <param name="page">The <see cref="Page"/> to render.</param>
01057 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the <see cref="Image"/>.</param>
01058 /// <returns>An <see cref="Image"/> containing the rasterised page.</returns>
01059         public static Image<SixLabors.ImageSharp.PixelFormats.Rgba32> SaveAsImage(this Page page,
      double scale = 1)
01060         {
01061             ImageSharpContext ctx = new ImageSharpContext(page.Width, page.Height, scale,
      page.Background);
01062             page.Graphics.CopyToIGraphicsContext(ctx);
01063             ctx.DisposeAllExceptImage();
01064             return ctx.Image;
01065         }
01066
01067
01068 /// <summary>
01069 /// Render the page to an image stream.
01070 /// </summary>
01071 /// <param name="page">The <see cref="Page"/> to render.</param>
01072 /// <param name="imageStream">The <see cref="Stream"/> on which the image data will be
      written.</param>
01073 /// <param name="outputFormat">The format of the image that will be created.</param>
01074 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01075         public static void SaveAsImage(this Page page, Stream imageStream, OutputFormats outputFormat,
      double scale = 1)
01076         {
01077             Image image = SaveAsImage(page, scale);
01078
01079             switch (outputFormat)
```

```
01080             {
01081                 case OutputFormats.BMP:
01082                     image.SaveAsBmp(imageStream);
01083                     break;
01084                 case OutputFormats.GIF:
01085                     image.SaveAsGif(imageStream);
01086                     break;
01087                 case OutputFormats.JPEG:
01088                     image.SaveAsJpeg(imageStream);
01089                     break;
01090                 case OutputFormats.PBM:
01091                     image.SaveAsPbm(imageStream);
01092                     break;
01093                 case OutputFormats.PNG:
01094                     image.SaveAsPng(imageStream);
01095                     break;
01096                 case OutputFormats.TGA:
01097                     image.SaveAsTga(imageStream);
01098                     break;
01099                 case OutputFormats.TIFF:
01100                     image.SaveAsTiff(imageStream);
01101                     break;
01102                 case OutputFormats.WebP:
01103                     image.SaveAsWebp(imageStream);
01104                     break;
01105             }
01106 
01107             image.Dispose();
01108         }
01109 
01110 /// <summary>
01111 /// The exception that is raised when the output file format is not specified and the file name does
      not have an extension corresponding to a known file format.
01112 /// </summary>
01113         public class UnknownFormatException :  Exception
01114         {
01115 /// <summary>
01116 /// The extension of the file that does not correspond to any known file format.
01117 /// </summary>
01118             public string Format { get; }
01119 
01120             internal UnknownFormatException(string format) :  base("The extension " + format + " does
      not correspond to any known file format!")
01121             {
01122                 this.Format = format;
01123             }
01124         }
01125 
01126 /// <summary>
01127 /// Render the page to an image file.
01128 /// </summary>
01129 /// <param name="page">The <see cref="Page"/> to render.</param>
01130 /// <param name="fileName">The path of the file where the image will be saved.</param>
01131 /// <param name="outputFormat">The format of the image that will be created.  If this is <see
      langword="null" /> (the default), the format is desumed from the extension of the file.</param>
01132 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01133         public static void SaveAsImage(this Page page, string fileName, OutputFormats?  outputFormat =
      null, double scale = 1)
01134         {
01135             OutputFormats actualOutputFormat;
01136 
01137             if (outputFormat.HasValue)
01138             {
01139                 actualOutputFormat = outputFormat.Value;
01140             }
01141             else
01142             {
01143                 string extension = System.IO.Path.GetExtension(fileName).ToLower();
01144 
01145                 switch (extension)
01146                 {
01147                     case ".bmp":
01148                     case ".dib":
01149                         actualOutputFormat = OutputFormats.BMP;
01150                         break;
01151 
01152                     case ".gif":
01153                         actualOutputFormat = OutputFormats.GIF;
01154                         break;
01155 
01156                     case ".jpeg":
01157                     case ".jpg":
01158                         actualOutputFormat = OutputFormats.JPEG;
01159                         break;
01160 
01161                     case ".pbm":
```

```
01162                         actualOutputFormat = OutputFormats.PBM;
01163                         break;
01164
01165                     case ".png":
01166                         actualOutputFormat = OutputFormats.PNG;
01167                         break;
01168
01169                     case ".tga":
01170                     case ".targa":
01171                         actualOutputFormat = OutputFormats.TGA;
01172                         break;
01173
01174                     case ".tif":
01175                     case ".tiff":
01176                         actualOutputFormat = OutputFormats.TIFF;
01177                         break;
01178
01179                     case ".webp":
01180                         actualOutputFormat = OutputFormats.WebP;
01181                         break;
01182
01183                     default:
01184                         throw new UnknownFormatException(extension);
01185                 }
01186             }
01187
01188             using (FileStream imageStream = new FileStream(fileName, FileMode.Create))
01189             {
01190                 SaveAsImage(page, imageStream, actualOutputFormat, scale);
01191             }
01192         }
01193
01194 /// <summary>
01195 /// Render the page to raw pixel data, in 32bpp RGBA format.
01196 /// </summary>
01197 /// <param name="pag">The <see cref="Page"/> to render.</param>
01198 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01199 /// <param name="width">The width of the rendered image.</param>
01200 /// <param name="height">The height of the rendered image.</param>
01201 /// <param name="totalSize">The size in bytes of the raw pixel data.</param>
01202 /// <returns>A <see cref="DisposableIntPtr"/> containing a pointer to the raw pixel data, stored in
      unmanaged memory.  Dispose this object to release the unmanaged memory.</returns>
01203         public static DisposableIntPtr SaveAsRawBytes(this Page pag, out int width, out int height,
      out int totalSize, double scale = 1)
01204         {
01205             Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01206
01207             int stride = img.Width * 4;
01208             int size = stride * img.Height;
01209
01210             IntPtr tbr = System.Runtime.InteropServices.Marshal.AllocHGlobal(size);
01211             GC.AddMemoryPressure(size);
01212
01213             IntPtr pointer = tbr;
01214
01215             unsafe
01216             {
01217                 for (int y = 0; y < img.Height; y++)
01218                 {
01219                     Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01220
01221                     Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
01222                     row.Span.CopyTo(newRow);
01223
01224                     pointer = IntPtr.Add(pointer, stride);
01225                 }
01226             }
01227
01228             width = img.Width;
01229             height = img.Height;
01230             totalSize = size;
01231
01232             img.Dispose();
01233
01234             return new DisposableIntPtr(tbr);
01235         }
01236
01237
01238 /// <summary>
01239 /// Return the page to raw pixel data, in 32bpp RGBA format.
01240 /// </summary>
01241 /// <param name="pag">The <see cref="Page"/> to render.</param>
01242 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
```

```
01243 /// <param name="width">The width of the rendered image.</param>
01244 /// <param name="height">The height of the rendered image.</param>
01245 /// <returns>A byte array containing the raw pixel data.</returns>
01246         public static byte[] SaveAsRawBytes(this Page pag, out int width, out int height, double scale
      = 1)
01247         {
01248             Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01249 
01250             int stride = img.Width * 4;
01251             int size = stride * img.Height;
01252 
01253             byte[] tbr = new byte[size];
01254 
01255             unsafe
01256             {
01257                 for (int y = 0; y < img.Height; y++)
01258                 {
01259                     Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01260 
01261                     Span<byte> bytes =
      System.Runtime.InteropServices.MemoryMarshal.Cast<SixLabors.ImageSharp.PixelFormats.Rgba32,
      byte>(row.Span);
01262                     Span<byte> newRow = new Span<byte>(tbr, y * stride, stride);
01263                     bytes.CopyTo(newRow);
01264                 }
01265             }
01266 
01267             width = img.Width;
01268             height = img.Height;
01269 
01270             img.Dispose();
01271 
01272             return tbr;
01273         }
01274 
01275 /// <summary>
01276 /// Rasterise a region of a <see cref="Graphics"/> object.
01277 /// </summary>
01278 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
01279 /// <param name="region">The region of the <paramref name="graphics"/> that will be
      rasterised.</param>
01280 /// <param name="scale">The scale at which the image will be rendered.</param>
01281 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is
      drawn on another <see cref="Graphics"/> surface.</param>
01282 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
01283         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale,
      bool interpolate)
01284         {
01285             Page pag = new Page(1, 1);
01286             pag.Graphics.DrawGraphics(0, 0, graphics);
01287             pag.Crop(region.Location, region.Size);
01288 
01289             Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01290 
01291             int stride = img.Width * 4;
01292             int size = stride * img.Height;
01293 
01294             IntPtr tbr = System.Runtime.InteropServices.Marshal.AllocHGlobal(size);
01295             GC.AddMemoryPressure(size);
01296 
01297             IntPtr pointer = tbr;
01298 
01299             unsafe
01300             {
01301                 for (int y = 0; y < img.Height; y++)
01302                 {
01303                     Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01304 
01305                     Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
01306                     row.Span.CopyTo(newRow);
01307 
01308                     pointer = IntPtr.Add(pointer, stride);
01309                 }
01310             }
01311 
01312             int width = img.Width;
01313             int height = img.Height;
01314 
01315             img.Dispose();
01316 
01317             DisposableIntPtr disp = new DisposableIntPtr(tbr);
01318 
01319             return new RasterImage(ref disp, width, height, true, interpolate);
01320         }
```

```
01321
01322 /// <summary>
01323 /// Saves the animation to an animated GIF.
01324 /// </summary>
01325 /// <param name="animation">The animation to export.</param>
01326 /// <param name="scale">The scale at which the animation will be rendered.</param>
01327 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated GIF specification at 50 fps.</param>
01328 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01329 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
      image, or if a different colour table should be used for each frame.</param>
01330 /// <returns>An <see cref="Image"/> containing the animated GIF.</returns>
01331         public static Image SaveAsAnimatedGIF(this Animation animation, double scale = 1, double
      frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
      GifColorTableMode.Local)
01332         {
01333             frameRate = Math.Min(frameRate, 50);
01334
01335             Image gifAnimation = new
      Image<SixLabors.ImageSharp.PixelFormats.Rgba32>((int)(animation.Width * scale), (int)(animation.Height
      * scale));
01336
01337             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
01338
01339             double accumulatedDelay = 0;
01340
01341             for (int i = 0; i < frames; i++)
01342             {
01343                 double frameTime = i / frameRate / durationScaling * 1000;
01344                 double frameDuration = Math.Min(animation.Duration - i / durationScaling / frameRate *
      1000, 1000 / frameRate / durationScaling);
01345
01346                 int gifFrameDuration = (int)Math.Round(frameDuration * durationScaling * 0.1 +
      accumulatedDelay);
01347
01348                 if (gifFrameDuration < 2)
01349                 {
01350                     accumulatedDelay += frameDuration * durationScaling * 0.1;
01351                 }
01352                 else
01353                 {
01354                     accumulatedDelay += frameDuration * durationScaling * 0.1 - gifFrameDuration;
01355
01356                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01357
01358                     Image frame = pag.SaveAsImage(scale);
01359
01360                     GifFrameMetadata frameMetadata =
      frame.Frames[0].Metadata.GetFormatMetadata(GifFormat.Instance);
01361                     frameMetadata.FrameDelay = gifFrameDuration;
01362                     frameMetadata.DisposalMethod = GifDisposalMethod.RestoreToPrevious;
01363                     gifAnimation.Frames.AddFrame(frame.Frames[0]);
01364                 }
01365             }
01366
01367             if ((int)accumulatedDelay > 0)
01368             {
01369                 gifAnimation.Frames[gifAnimation.Frames.Count -
      1].Metadata.GetFormatMetadata(GifFormat.Instance).FrameDelay += (int)accumulatedDelay;
01370             }
01371
01372             gifAnimation.Frames.RemoveFrame(0);
01373
01374             GifMetadata metadata = gifAnimation.Metadata.GetFormatMetadata(GifFormat.Instance);
01375             metadata.ColorTableMode = colorTableMode;
01376             metadata.RepeatCount = (ushort)animation.RepeatCount;
01377
01378             return gifAnimation;
01379         }
01380
01381 /// <summary>
01382 /// Saves the animation to an animated GIF stream.
01383 /// </summary>
01384 /// <param name="animation">The animation to export.</param>
01385 /// <param name="imageStream">The stream on which the animated GIF will be written.</param>
01386 /// <param name="scale">The scale at which the animation will be rendered.</param>
01387 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated GIF specification at 50 fps.</param>
01388 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01389 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
      image, or if a different colour table should be used for each frame.</param>
01390         public static void SaveAsAnimatedGIF(this Animation animation, Stream imageStream, double
      scale = 1, double frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
```

```
    GifColorTableMode.Local)
01391         {
01392             Image gifAnimation = SaveAsAnimatedGIF(animation, scale, frameRate, durationScaling,
    colorTableMode);
01393 
01394             gifAnimation.SaveAsGif(imageStream);
01395         }
01396 
01397 /// <summary>
01398 /// Saves the animation to an animated GIF file.
01399 /// </summary>
01400 /// <param name="animation">The animation to export.</param>
01401 /// <param name="fileName">The output file to create.</param>
01402 /// <param name="scale">The scale at which the animation will be rendered.</param>
01403 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
    is capped by the animated GIF specification at 50 fps.</param>
01404 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
    animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
    that this does not affect the frame rate of the animation.</param>
01405 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
    image, or if a different colour table should be used for each frame.</param>
01406         public static void SaveAsAnimatedGIF(this Animation animation, string fileName, double scale =
    1, double frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
    GifColorTableMode.Local)
01407         {
01408             using (FileStream fs = File.Create(fileName))
01409             {
01410                 SaveAsAnimatedGIF(animation, fs, scale, frameRate, durationScaling, colorTableMode);
01411             }
01412         }
01413 
01414 /// <summary>
01415 /// Saves the animation to a stream in animated PNG format.
01416 /// </summary>
01417 /// <param name="animation">The animation to export.</param>
01418 /// <param name="imageStream">The stream on which the animated PNG will be written.</param>
01419 /// <param name="scale">The scale at which the animation will be rendered.</param>
01420 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
    is capped by the animated PNG specification at 90 fps.</param>
01421 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
    animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
    that this does not affect the frame rate of the animation.</param>
01422 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
    Note that if the animation has a transparent background, no compression can be performed, and the
    value of this parameter is ignored.</param>
01423         public static void SaveAsAnimatedPNG(this Animation animation, Stream imageStream, double
    scale = 1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
    interframeCompression = AnimatedPNG.InterframeCompression.First)
01424         {
01425             frameRate = Math.Min(frameRate, 90);
01426 
01427             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
01428 
01429             Stream fs = imageStream;
01430 
01431             AnimatedPNG.CompressedFrame[] compressedFrames = new AnimatedPNG.CompressedFrame[frames];
01432 
01433             int width = (int)(animation.Width * scale);
01434             int height = (int)(animation.Height * scale);
01435 
01436             if (animation.Background.A < 1 || interframeCompression ==
    AnimatedPNG.InterframeCompression.None)
01437             {
01438                 Parallel.For(0, frames, i =>
01439                 {
01440                     double frameTime = i / frameRate / durationScaling * 1000;
01441 
01442                     double frameDuration = Math.Min((animation.Duration - frameTime) *
    durationScaling, 1000 / frameRate);
01443 
01444                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01445 
01446                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
01447                     {
01448                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, width, height,
    true, frameDuration);
01449                     }
01450                 });
01451             }
01452             else if (interframeCompression == AnimatedPNG.InterframeCompression.First)
01453             {
01454                 DisposableIntPtr firstFrame = animation.GetFrameAtAbsolute(0).SaveAsRawBytes(out _,
    out _, out _, scale);
01455                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(firstFrame, width, height, true,
    Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
01456 
01457                 Parallel.For(1, frames, i =>
```

```
01458                 {
01459                     double frameTime = i / frameRate / durationScaling * 1000;
01460 
01461                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
01462 
01463                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01464 
01465                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
01466                     {
01467                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, firstFrame,
      true, width, height, true, frameDuration);
01468                     }
01469                 });
01470 
01471                 firstFrame.Dispose();
01472             }
01473             else if (interframeCompression == AnimatedPNG.InterframeCompression.Previous)
01474             {
01475                 DisposableIntPtr[] rawFrames = new DisposableIntPtr[frames];
01476 
01477                 Parallel.For(0, frames, i =>
01478                 {
01479                     double frameTime = i / frameRate / durationScaling * 1000;
01480                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01481                     rawFrames[i] = pag.SaveAsRawBytes(out _, out _, out _, scale);
01482                 });
01483 
01484                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(rawFrames[0], width, height,
      true, Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
01485 
01486                 Parallel.For(1, frames, i =>
01487                 {
01488                     double frameTime = i / frameRate / durationScaling * 1000;
01489                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
01490                     compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrames[i], rawFrames[i -
      1], false, width, height, true, frameDuration);
01491                 });
01492             }
01493 
01494             AnimatedPNG.Create(fs, (int)(animation.Width * scale), (int)(animation.Height * scale),
      true, compressedFrames, animation.RepeatCount);
01495 
01496             for (int i = 0; i < compressedFrames.Length; i++)
01497             {
01498                 compressedFrames[i].Dispose();
01499             }
01500         }
01501 
01502 /// <summary>
01503 /// Saves the animation to an animated PNG file.
01504 /// </summary>
01505 /// <param name="animation">The animation to export.</param>
01506 /// <param name="fileName">The output file to create.</param>
01507 /// <param name="scale">The scale at which the animation will be rendered.</param>
01508 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
01509 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01510 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
01511         public static void SaveAsAnimatedPNG(this Animation animation, string fileName, double scale =
      1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
01512         {
01513             using (FileStream fs = File.Create(fileName))
01514             {
01515                 SaveAsAnimatedPNG(animation, fs, scale, frameRate, durationScaling,
      interframeCompression);
01516             }
01517         }
01518     }
01519 }
```

8.43 Raster.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004 
```

```
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Runtime.InteropServices;
00022 using System.Threading.Tasks;
00023 using VectSharp.PDF;
00024
00025 namespace VectSharp.Raster
00026 {
00027 /// <summary>
00028 /// Contains methods to render a page to a PNG image.
00029 /// </summary>
00030     public static class Raster
00031     {
00032
00033 /// <summary>
00034 /// Render the page to a PNG file.
00035 /// </summary>
00036 /// <param name="page">The <see cref="Page"/> to render.</param>
00037 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
00038 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image file.</param>
00039         public static void SaveAsPNG(this Page page, string fileName, double scale = 1)
00040         {
00041             Document doc = new Document();
00042             doc.Pages.Add(page);
00043
00044             MemoryStream ms = new MemoryStream();
00045             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00046
00047             using (MuPDFContext context = new MuPDFContext())
00048             {
00049                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00050                 {
00051                     muDoc.SaveImage(0, scale, MuPDFCore.PixelFormats.RGBA, fileName,
      RasterOutputFileTypes.PNG);
00052                 }
00053             }
00054
00055             ms.Dispose();
00056         }
00057
00058 /// <summary>
00059 /// Render the page to a PNG stream.
00060 /// </summary>
00061 /// <param name="page">The <see cref="Page"/> to render.</param>
00062 /// <param name="stream">The stream to which the PNG data will be written.</param>
00063 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image file.</param>
00064         public static void SaveAsPNG(this Page page, Stream stream, double scale = 1)
00065         {
00066             Document doc = new Document();
00067             doc.Pages.Add(page);
00068
00069             MemoryStream ms = new MemoryStream();
00070             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00071
00072             using (MuPDFContext context = new MuPDFContext())
00073             {
00074                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00075                 {
00076                     muDoc.WriteImage(0, scale, MuPDFCore.PixelFormats.RGBA, stream,
      RasterOutputFileTypes.PNG);
00077                 }
00078             }
00079
00080             ms.Dispose();
```

```
00081         }
00082
00083 /// <summary>
00084 /// Rasterise a region of a <see cref="Graphics"/> object.
00085 /// </summary>
00086 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
00087 /// <param name="region">The region of the <paramref name="graphics"/> that will be
      rasterised.</param>
00088 /// <param name="scale">The scale at which the image will be rendered.</param>
00089 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is
      drawn on another <see cref="Graphics"/> surface.</param>
00090 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
00091         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale,
      bool interpolate)
00092         {
00093             Page pag = new Page(1, 1);
00094             pag.Graphics.DrawGraphics(0, 0, graphics);
00095             pag.Crop(region.Location, region.Size);
00096
00097             Document doc = new Document();
00098             doc.Pages.Add(pag);
00099
00100             MemoryStream ms = new MemoryStream();
00101             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00102
00103             IntPtr imageDataAddress;
00104
00105             int width;
00106             int height;
00107
00108             using (MuPDFContext context = new MuPDFContext())
00109             {
00110                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00111                 {
00112                     int imageSize = muDoc.GetRenderedSize(0, scale, MuPDFCore.PixelFormats.RGBA);
00113
00114                     RoundedRectangle roundedBounds = muDoc.Pages[0].Bounds.Round(scale);
00115
00116                     width = roundedBounds.Width;
00117                     height = roundedBounds.Height;
00118
00119
00120                     imageDataAddress = Marshal.AllocHGlobal(imageSize);
00121                     GC.AddMemoryPressure(imageSize);
00122
00123                     muDoc.Render(0, scale, MuPDFCore.PixelFormats.RGBA, imageDataAddress);
00124                 }
00125             }
00126
00127             ms.Dispose();
00128
00129             DisposableIntPtr disposableAddress = new DisposableIntPtr(imageDataAddress);
00130
00131             return new RasterImage(ref disposableAddress, width, height, true, interpolate);
00132         }
00133
00134 /// <summary>
00135 /// Render the page to raw pixel data, in 32bpp RGBA format.
00136 /// </summary>
00137 /// <param name="pag">The <see cref="Page"/> to render.</param>
00138 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
00139 /// <param name="width">The width of the rendered image.</param>
00140 /// <param name="height">The height of the rendered image.</param>
00141 /// <param name="totalSize">The size in bytes of the raw pixel data.</param>
00142 /// <returns>A <see cref="DisposableIntPtr"/> containing a pointer to the raw pixel data, stored in
      unmanaged memory.  Dispose this object to release the unmanaged memory.</returns>
00143         public static DisposableIntPtr SaveAsRawBytes(this Page pag, out int width, out int height,
      out int totalSize, double scale = 1)
00144         {
00145             Document doc = new Document();
00146             doc.Pages.Add(pag);
00147
00148             MemoryStream ms = new MemoryStream();
00149             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00150
00151             IntPtr renderedPage;
00152
00153             using (MuPDFContext context = new MuPDFContext())
00154             {
00155                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
```

```
00156                 {
00157                     totalSize = muDoc.GetRenderedSize(0, scale, MuPDFCore.PixelFormats.RGBA);
00158 
00159                     renderedPage = Marshal.AllocHGlobal(totalSize);
00160 
00161                     RoundedRectangle bounds = muDoc.Pages[0].Bounds.Round(scale);
00162 
00163                     width = bounds.Width;
00164                     height = bounds.Height;
00165 
00166                     muDoc.Render(0, scale, MuPDFCore.PixelFormats.RGBA, renderedPage);
00167                 }
00168             }
00169 
00170             ms.Dispose();
00171 
00172             return new DisposableIntPtr(renderedPage);
00173         }
00174 
00175 /// <summary>
00176 /// Saves the animation to a stream in animated PNG format.
00177 /// </summary>
00178 /// <param name="animation">The animation to export.</param>
00179 /// <param name="imageStream">The stream on which the animated PNG will be written.</param>
00180 /// <param name="scale">The scale at which the animation will be rendered.</param>
00181 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
00182 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
00183 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
00184         public static void SaveAsAnimatedPNG(this Animation animation, Stream imageStream, double
      scale = 1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
00185         {
00186             frameRate = Math.Min(frameRate, 90);
00187 
00188             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
00189 
00190             Stream fs = imageStream;
00191 
00192             AnimatedPNG.CompressedFrame[] compressedFrames = new AnimatedPNG.CompressedFrame[frames];
00193 
00194             int width = (int)(animation.Width * scale);
00195             int height = (int)(animation.Height * scale);
00196 
00197             if (animation.Background.A < 1 || interframeCompression ==
      AnimatedPNG.InterframeCompression.None)
00198             {
00199                 Parallel.For(0, frames, i =>
00200                 {
00201                     double frameTime = i / frameRate / durationScaling * 1000;
00202 
00203                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
00204 
00205                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00206 
00207                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
00208                     {
00209                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, width, height,
      true, frameDuration);
00210                     }
00211                 });
00212             }
00213             else if (interframeCompression == AnimatedPNG.InterframeCompression.First)
00214             {
00215                 DisposableIntPtr firstFrame = animation.GetFrameAtAbsolute(0).SaveAsRawBytes(out _,
      out _, out _, scale);
00216                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(firstFrame, width, height, true,
      Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
00217 
00218                 Parallel.For(1, frames, i =>
00219                 {
00220                     double frameTime = i / frameRate / durationScaling * 1000;
00221 
00222                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
00223 
00224                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00225 
00226                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
00227                     {
00228                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, firstFrame,
      true, width, height, true, frameDuration);
```

```
00229                     }
00230                 });
00231 
00232                 firstFrame.Dispose();
00233             }
00234             else if (interframeCompression == AnimatedPNG.InterframeCompression.Previous)
00235             {
00236                 DisposableIntPtr[] rawFrames = new DisposableIntPtr[frames];
00237 
00238                 Parallel.For(0, frames, i =>
00239                 {
00240                     double frameTime = i / frameRate / durationScaling * 1000;
00241                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00242                     rawFrames[i] = pag.SaveAsRawBytes(out _, out _, out _, scale);
00243                 });
00244 
00245                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(rawFrames[0], width, height,
    true, Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
00246 
00247                 Parallel.For(1, frames, i =>
00248                 {
00249                     double frameTime = i / frameRate / durationScaling * 1000;
00250                     double frameDuration = Math.Min((animation.Duration - frameTime) *
    durationScaling, 1000 / frameRate);
00251                     compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrames[i], rawFrames[i -
    1], false, width, height, true, frameDuration);
00252                 });
00253             }
00254 
00255             AnimatedPNG.Create(fs, (int)(animation.Width * scale), (int)(animation.Height * scale),
    true, compressedFrames, animation.RepeatCount);
00256         }
00257 
00258 /// <summary>
00259 /// Saves the animation to an animated PNG file.
00260 /// </summary>
00261 /// <param name="animation">The animation to export.</param>
00262 /// <param name="fileName">The output file to create.</param>
00263 /// <param name="scale">The scale at which the animation will be rendered.</param>
00264 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
    is capped by the animated PNG specification at 90 fps.</param>
00265 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
    animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
    that this does not affect the frame rate of the animation.</param>
00266 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
    Note that if the animation has a transparent background, no compression can be performed, and the
    value of this parameter is ignored.</param>
00267         public static void SaveAsAnimatedPNG(this Animation animation, string fileName, double scale =
    1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
    interframeCompression = AnimatedPNG.InterframeCompression.First)
00268         {
00269             using (FileStream fs = File.Create(fileName))
00270             {
00271                 SaveAsAnimatedPNG(animation, fs, scale, frameRate, durationScaling,
    interframeCompression);
00272             }
00273         }
00274     }
00275 }
```

8.44 SVGContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using ExCSS;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Data;
```

```
00022 using System.IO;
00023 using System.Linq;
00024 using System.Reflection;
00025 using System.Text;
00026 using System.Threading.Tasks;
00027 using System.Transactions;
00028 using System.Xml;
00029 using VectSharp.Filters;
00030
00031 namespace VectSharp.SVG
00032 {
00033     internal static class MatrixUtils
00034     {
00035         public static double[] Multiply(double[,] matrix, double[] vector)
00036         {
00037             double[] tbr = new double[2];
00038
00039             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00040             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00041
00042             return tbr;
00043         }
00044
00045         public static Point Multiply(double[,] matrix, Point vector)
00046         {
00047             double[] tbr = new double[2];
00048
00049             tbr[0] = matrix[0, 0] * vector.X + matrix[0, 1] * vector.Y + matrix[0, 2];
00050             tbr[1] = matrix[1, 0] * vector.X + matrix[1, 1] * vector.Y + matrix[1, 2];
00051
00052             return new Point(tbr[0], tbr[1]);
00053         }
00054
00055         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00056         {
00057             double[,] tbr = new double[3, 3];
00058
00059             for (int i = 0; i < 3; i++)
00060             {
00061                 for (int j = 0; j < 3; j++)
00062                 {
00063                     for (int k = 0; k < 3; k++)
00064                     {
00065                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00066                     }
00067                 }
00068             }
00069
00070             return tbr;
00071         }
00072
00073         public static double[,] Rotate(double[,] matrix, double angle)
00074         {
00075             double[,] rotationMatrix = new double[3, 3];
00076             rotationMatrix[0, 0] = Math.Cos(angle);
00077             rotationMatrix[0, 1] = -Math.Sin(angle);
00078             rotationMatrix[1, 0] = Math.Sin(angle);
00079             rotationMatrix[1, 1] = Math.Cos(angle);
00080             rotationMatrix[2, 2] = 1;
00081
00082             return Multiply(matrix, rotationMatrix);
00083         }
00084
00085         public static double[,] Translate(double[,] matrix, double x, double y)
00086         {
00087             double[,] translationMatrix = new double[3, 3];
00088             translationMatrix[0, 0] = 1;
00089             translationMatrix[0, 2] = x;
00090             translationMatrix[1, 1] = 1;
00091             translationMatrix[1, 2] = y;
00092             translationMatrix[2, 2] = 1;
00093
00094             return Multiply(matrix, translationMatrix);
00095         }
00096
00097         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00098         {
00099             double[,] scaleMatrix = new double[3, 3];
00100             scaleMatrix[0, 0] = scaleX;
00101             scaleMatrix[1, 1] = scaleY;
00102             scaleMatrix[2, 2] = 1;
00103
00104             return Multiply(matrix, scaleMatrix);
00105         }
00106
00107         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00108
```

```
00109         public static double[,] Invert(double[,] m)
00110         {
00111             double[,] tbr = new double[3, 3];
00112 
00113             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00114             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00115             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00116             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00117             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00118             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00119             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00120             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00121             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00122 
00123             return tbr;
00124         }
00125 
00126         public static Func<Point, Point> GetInverseTransformation(double[,] m)
00127         {
00128             double[,] inverted = Invert(m);
00129 
00130             return x => Multiply(inverted, x);
00131         }
00132     }
00133 
00134     internal class SVGFigure
00135     {
00136         public Point StartPoint { get; set; }
00137         public Point CurrentPoint { get; set; }
00138 
00139         public string Data { get; set; }
00140         public int PointCount { get; set; } = 0;
00141     }
00142 
00143     internal class SVGPathObject
00144     {
00145         public List<SVGFigure> Figures { get; set; } = new List<SVGFigure>();
00146     }
00147 
00148 
00149     internal class SVGContext :  IGraphicsContext
00150     {
00151         public Dictionary<string, FontFamily> UsedFontFamilies;
00152         public Dictionary<string, HashSet<char» UsedChars;
00153 
00154 
00155         public const string SVGNamespace = "http://www.w3.org/2000/svg";
00156 
00157         public double Width { get; }
00158 
00159         public double Height { get; }
00160 
00161         public Font Font { get; set; }
00162         public TextBaselines TextBaseline { get; set; }
00163 
00164         public Brush FillStyle { get; private set; }
00165 
00166         public Brush StrokeStyle { get; private set; }
00167 
00168         public double LineWidth { get; set; }
00169         public LineCaps LineCap { private get; set; }
00170         public LineJoins LineJoin { private get; set; }
00171 
00172         private LineDash _lineDash;
00173 
00174         private SVGPathObject currentPath;
00175         private SVGFigure currentFigure;
00176 
00177         public XmlDocument Document;
```

```
00178         private XmlElement currentElement;
00179
00180         public string Tag { get; set; }
00181         public string TagPrefix { get; }
00182
00183         private bool ReuseGradients { get; }
00184
00185         private double[,] _transform;
00186         private Stack<double[,]> states;
00187
00188         private string _currClipPath;
00189         private Stack<string> clipPaths;
00190
00191         private bool TextToPaths = false;
00192         private SVGContextInterpreter.TextOptions TextOption =
      SVGContextInterpreter.TextOptions.SubsetFonts;
00193
00194         private Dictionary<string, string> linkDestinations;
00195
00196         XmlElement definitions;
00197         Dictionary<Brush, string> gradients;
00198         int gradientCount = 0;
00199
00200         private SVGContextInterpreter.FilterOption FilterOption;
00201
00202         private bool UseStyles;
00203         internal Dictionary<string, Dictionary<string, string>> Styles;
00204         XmlElement StyleElement;
00205
00206         private string GetClass(Dictionary<string, string> style)
00207         {
00208             foreach (KeyValuePair<string, Dictionary<string, string>> styleClass in Styles)
00209             {
00210                 if (styleClass.Value.Count == style.Count)
00211                 {
00212                     bool mismatch = false;
00213
00214                     foreach (KeyValuePair<string, string> styleItem in style)
00215                     {
00216                         if (!styleClass.Value.TryGetValue(styleItem.Key, out string value))
00217                         {
00218                             mismatch = true;
00219                             break;
00220                         }
00221                         else if (value != styleItem.Value)
00222                         {
00223                             mismatch = true;
00224                             break;
00225                         }
00226                     }
00227
00228                     if (!mismatch)
00229                     {
00230                         return styleClass.Key;
00231                     }
00232                 }
00233             }
00234
00235             string tbr = "class" + (Styles.Count + 1).ToString();
00236
00237             Styles.Add(tbr, style);
00238
00239             return tbr;
00240         }
00241
00242         public SVGContext(double width, double height, bool textToPaths,
      SVGContextInterpreter.TextOptions textOption, Dictionary<string, string> linkDestinations,
      SVGContextInterpreter.FilterOption filterOption, bool useStyles, string tagPrefix, bool
      reuseGradients)
00243         {
00244             this.linkDestinations = linkDestinations;
00245
00246             this.TagPrefix = tagPrefix;
00247
00248             this.Width = width;
00249             this.Height = height;
00250
00251             currentPath = new SVGPathObject();
00252             currentFigure = new SVGFigure();
00253
00254             StrokeStyle = Colour.FromRgba(0, 0, 0, 0);
00255             FillStyle = Colour.FromRgb(0, 0, 0);
00256             LineWidth = 1;
00257
00258             LineCap = LineCaps.Butt;
00259             LineJoin = LineJoins.Miter;
00260             _lineDash = new LineDash(0, 0, 0);
```

```
00261
00262             Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica),
      12);
00263
00264             TextBaseline = TextBaselines.Top;
00265
00266             _transform = new double[3, 3];
00267
00268             _transform[0, 0] = 1;
00269             _transform[1, 1] = 1;
00270             _transform[2, 2] = 1;
00271
00272             states = new Stack<double[,]>();
00273
00274             clipPaths = new Stack<string>();
00275             clipPaths.Push(null);
00276             _currClipPath = null;
00277
00278             UsedFontFamilies = new Dictionary<string, FontFamily>();
00279             UsedChars = new Dictionary<string, HashSet<char»();
00280
00281             Document = new XmlDocument();
00282
00283             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null),
      Document.DocumentElement);
00284
00285             currentElement = Document.CreateElement(null, "svg", SVGNamespace);
00286             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
00287             currentElement.SetAttribute("viewBox", "0 0 " +
      width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      height.ToString(System.Globalization.CultureInfo.InvariantCulture));
00288             currentElement.SetAttribute("version", "1.1");
00289             Document.AppendChild(currentElement);
00290
00291             definitions = Document.CreateElement("defs", SVGNamespace);
00292             gradients = new Dictionary<Brush, string>();
00293             currentElement.AppendChild(definitions);
00294
00295             this.UseStyles = useStyles;
00296             StyleElement = Document.CreateElement("style", SVGNamespace);
00297             StyleElement.InnerText = "\t";
00298             currentElement.AppendChild(StyleElement);
00299             Styles = new Dictionary<string, Dictionary<string, string»();
00300
00301             this.TextToPaths = textToPaths;
00302             this.TextOption = textOption;
00303             this.FilterOption = filterOption;
00304             this.ReuseGradients = reuseGradients;
00305         }
00306
00307
00308         private void PathText(string text, double x, double y)
00309         {
00310             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00311
00312             for (int j = 0; j < textPath.Segments.Count; j++)
00313             {
00314                 switch (textPath.Segments[j].Type)
00315                 {
00316                     case VectSharp.SegmentType.Move:
00317                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00318                         break;
00319                     case VectSharp.SegmentType.Line:
00320                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00321                         break;
00322                     case VectSharp.SegmentType.CubicBezier:
00323                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00324                         break;
00325                     case VectSharp.SegmentType.Close:
00326                         this.Close();
00327                         break;
00328                 }
00329             }
00330         }
00331
00332         public void Close()
00333         {
00334             if (currentFigure.PointCount > 0)
00335             {
00336                 currentFigure.Data += "Z ";
00337                 currentPath.Figures.Add(currentFigure);
00338
00339                 currentFigure = new SVGFigure();
00340             }
00341         }
```

```
00342
00343         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00344         {
00345             if (currentFigure.PointCount == 0)
00346             {
00347                 currentFigure.StartPoint = new Point(p1X, p1Y);
00348             }
00349
00350             currentFigure.CurrentPoint = new Point(p3X, p3Y);
00351             currentFigure.Data += "C " +
      p1X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p1Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", " +
00352                 p2X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p2Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", " +
00353                 p3X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p3Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00354             currentFigure.PointCount += 3;
00355         }
00356
00357         public void Fill(FillRule fillRule)
00358         {
00359             if (currentFigure.PointCount > 0)
00360             {
00361                 currentPath.Figures.Add(currentFigure);
00362             }
00363
00364             XmlElement currElement = currentElement;
00365
00366             if (!string.IsNullOrEmpty(_currClipPath))
00367             {
00368                 currentElement = Document.CreateElement("g", SVGNamespace);
00369                 currentElement.SetAttribute("clip-path", _currClipPath);
00370                 currElement.AppendChild(currentElement);
00371             }
00372
00373             XmlElement path = Document.CreateElement("path", SVGNamespace);
00374             path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
00375
00376             string gradientName = null;
00377
00378             {
00379                 if (FillStyle is LinearGradientBrush linearGradient)
00380                 {
00381                     if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
00382                     {
00383                         if (!string.IsNullOrEmpty(Tag))
00384                         {
00385                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00386                         }
00387                         else
00388                         {
00389                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00390                         }
00391
00392                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
00393                         this.definitions.AppendChild(gradientElement);
00394
00395                         if (ReuseGradients)
00396                         {
00397                             gradients.Add(linearGradient, gradientName);
00398                         }
00399
00400                         gradientCount++;
00401                     }
00402                 }
00403                 else if (FillStyle is RadialGradientBrush radialGradient)
00404                 {
00405                     if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
00406                     {
00407                         if (!string.IsNullOrEmpty(Tag))
00408                         {
00409                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00410                         }
00411                         else
00412                         {
00413                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00414                         }
00415
00416                         XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
00417                         this.definitions.AppendChild(gradientElement);
```

```
00418
00419                           if (ReuseGradients)
00420                           {
00421                               gradients.Add(radialGradient, gradientName);
00422                           }
00423
00424                           gradientCount++;
00425                       }
00426                   }
00427               }
00428
00429               if (!UseStyles)
00430               {
00431                   path.SetAttribute("stroke", "none");
00432
00433                   switch (fillRule)
00434                   {
00435                       case FillRule.EvenOdd:
00436                           path.SetAttribute("fill-rule", "evenodd");
00437                           break;
00438
00439                       case FillRule.NonZeroWinding:
00440                           path.SetAttribute("fill-rule", "nonzero");
00441                           break;
00442                   }
00443
00444                   if (FillStyle is SolidColourBrush solid)
00445                   {
00446                       path.SetAttribute("fill", solid.Colour.ToCSSString(false));
00447                       path.SetAttribute("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00448                   }
00449                   else if (FillStyle is LinearGradientBrush linearGradient)
00450                   {
00451                       path.SetAttribute("fill", "url(#" + gradientName + ")");
00452                   }
00453                   else if (FillStyle is RadialGradientBrush radialGradient)
00454                   {
00455                       path.SetAttribute("fill", "url(#" + gradientName + ")");
00456                   }
00457
00458                   path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00459                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00460                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00461               }
00462               else
00463               {
00464                   Dictionary<string, string> style = new Dictionary<string, string>();
00465
00466                   style.Add("stroke", "none");
00467
00468                   switch (fillRule)
00469                   {
00470                       case FillRule.EvenOdd:
00471                           style.Add("fill-rule", "evenodd");
00472                           break;
00473
00474                       case FillRule.NonZeroWinding:
00475                           style.Add("fill-rule", "nonzero");
00476                           break;
00477                   }
00478
00479                   if (FillStyle is SolidColourBrush solid)
00480                   {
00481                       style.Add("fill", solid.Colour.ToCSSString(false));
00482                       style.Add("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00483                   }
00484                   else if (FillStyle is LinearGradientBrush linearGradient)
00485                   {
00486                       style.Add("fill", "url(#" + gradientName + ")");
00487                   }
00488                   else if (FillStyle is RadialGradientBrush radialGradient)
00489                   {
00490                       style.Add("fill", "url(#" + gradientName + ")");
00491                   }
00492
00493                   style.Add("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00494                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00495                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
```

```
    + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00496
00497                   string className = GetClass(style);
00498
00499                   path.SetAttribute("class", className);
00500               }
00501
00502               if (!string.IsNullOrEmpty(Tag))
00503               {
00504                   path.SetAttribute("id", this.TagPrefix + Tag);
00505               }
00506
00507               if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
    string destination) && !string.IsNullOrEmpty(destination))
00508               {
00509                   XmlElement aElement = Document.CreateElement("a", SVGNamespace);
00510                   aElement.SetAttribute("href", destination);
00511                   currentElement.AppendChild(aElement);
00512                   currentElement = aElement;
00513               }
00514
00515               currentElement.AppendChild(path);
00516
00517               currentElement = currElement;
00518
00519               currentPath = new SVGPathObject();
00520               currentFigure = new SVGFigure();
00521
00522           }
00523
00524           public void FillText(string text, double x, double y)
00525           {
00526               if ((!TextToPaths && TextOption !=
    SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs) || (TextOption ==
    SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs && FillStyle is SolidColourBrush))
00527               {
00528                   if (!UsedFontFamilies.ContainsKey(Font.FontFamily.FileName))
00529                   {
00530                       UsedFontFamilies.Add(Font.FontFamily.FileName, Font.FontFamily);
00531                       UsedChars.Add(Font.FontFamily.FileName, new HashSet<char>());
00532                   }
00533
00534                   UsedChars[Font.FontFamily.FileName].UnionWith(text);
00535
00536                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00537
00538                   double[,] currTransform = null;
00539                   double[,] deltaTransform = MatrixUtils.Identity;
00540
00541                   switch (TextBaseline)
00542                   {
00543                       case TextBaselines.Baseline:
00544                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
    y);
00545                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
    metrics.LeftSideBearing, y);
00546                           break;
00547                       case TextBaselines.Top:
00548                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
    y + metrics.Top);
00549                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
    metrics.LeftSideBearing, y + metrics.Top);
00550                           break;
00551                       case TextBaselines.Bottom:
00552                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
    y + metrics.Bottom);
00553                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
    metrics.LeftSideBearing, y + metrics.Bottom);
00554                           break;
00555                       case TextBaselines.Middle:
00556                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
    y + (metrics.Top + metrics.Bottom) * 0.5);
00557                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
    metrics.LeftSideBearing, y + (metrics.Top + metrics.Bottom) * 0.5);
00558                           break;
00559                       default:
00560                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
    y);
00561                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
    metrics.LeftSideBearing, y);
00562                           break;
00563                   }
00564
00565                   XmlElement currElement = currentElement;
00566
00567                   if (!string.IsNullOrEmpty(_currClipPath))
00568                   {
```

```
00569                       currentElement = Document.CreateElement("g", SVGNamespace);
00570                       currentElement.SetAttribute("clip-path", _currClipPath);
00571                       currElement.AppendChild(currentElement);
00572                   }
00573
00574                   XmlElement textElement;
00575
00576                   if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
00577                   {
00578                       textElement = Document.CreateElement("text", SVGNamespace);
00579                   }
00580                   else
00581                   {
00582                       textElement = Document.CreateElement("g", SVGNamespace);
00583                   }
00584
00585
00586                   string gradientName = null;
00587
00588                   {
00589                       if (FillStyle is LinearGradientBrush linearGradient)
00590                       {
00591                           if (!string.IsNullOrEmpty(Tag))
00592                           {
00593                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00594                           }
00595                           else
00596                           {
00597                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00598                           }
00599
00600                           XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
00601
00602                           deltaTransform = MatrixUtils.Invert(deltaTransform);
00603
00604                           gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00605                           "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00606                           "," + deltaTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00607
00608                           this.definitions.AppendChild(gradientElement);
00609
00610                           gradientCount++;
00611                       }
00612                       else if (FillStyle is RadialGradientBrush radialGradient)
00613                       {
00614                           if (!string.IsNullOrEmpty(Tag))
00615                           {
00616                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00617                           }
00618                           else
00619                           {
00620                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00621                           }
00622
00623                           XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
00624
00625                           deltaTransform = MatrixUtils.Invert(deltaTransform);
00626
00627                           gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00628                           "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00629                           "," + deltaTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00630
00631                           this.definitions.AppendChild(gradientElement);
00632
00633                           gradientCount++;
00634                       }
00635                   }
00636
00637                   if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
```

```
00638                 {
00639                     textElement.SetAttribute("x", "0");
00640                     textElement.SetAttribute("y", "0");
00641                 }
00642 
00643                 textElement.SetAttribute("transform", "matrix(" + currTransform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00644                     "," + currTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00645                     "," + currTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00646 
00647                 if (!UseStyles)
00648                 {
00649                     textElement.SetAttribute("stroke", "none");
00650 
00651                     if (FillStyle is SolidColourBrush solid)
00652                     {
00653                         textElement.SetAttribute("fill", solid.Colour.ToCSSString(false));
00654                         textElement.SetAttribute("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00655                     }
00656                     else if (FillStyle is LinearGradientBrush linearGradient)
00657                     {
00658                         textElement.SetAttribute("fill", "url(#" + gradientName + ")");
00659                     }
00660                     else if (FillStyle is RadialGradientBrush radialGradient)
00661                     {
00662                         textElement.SetAttribute("fill", "url(#" + gradientName + ")");
00663                     }
00664 
00665                     textElement.SetAttribute("font-size",
      Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture));
00666                     textElement.SetAttribute("font-family", Font.FontFamily.FileName);
00667 
00668                     if (Font.FontFamily.IsBold)
00669                     {
00670                         textElement.SetAttribute("font-weight", "bold");
00671                     }
00672                     else
00673                     {
00674                         textElement.SetAttribute("font-weight", "regular");
00675                     }
00676 
00677                     if (Font.FontFamily.IsItalic)
00678                     {
00679                         textElement.SetAttribute("font-style", "italic");
00680                     }
00681                     else
00682                     {
00683                         textElement.SetAttribute("font-style", "normal");
00684                     }
00685 
00686                     if (Font.FontFamily.IsOblique)
00687                     {
00688                         textElement.SetAttribute("font-style", "oblique");
00689                     }
00690                 }
00691                 else
00692                 {
00693                     Dictionary<string, string> style = new Dictionary<string, string>();
00694 
00695                     style.Add("stroke", "none");
00696 
00697                     if (FillStyle is SolidColourBrush solid)
00698                     {
00699                         style.Add("fill", solid.Colour.ToCSSString(false));
00700                         style.Add("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00701                     }
00702                     else if (FillStyle is LinearGradientBrush linearGradient)
00703                     {
00704                         style.Add("fill", "url(#" + gradientName + ")");
00705                     }
00706                     else if (FillStyle is RadialGradientBrush radialGradient)
00707                     {
00708                         style.Add("fill", "url(#" + gradientName + ")");
00709                     }
00710 
00711                     style.Add("font-size",
      Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
00712                     style.Add("font-family", Font.FontFamily.FileName);
00713 
00714                     if (Font.FontFamily.IsBold)
```

```
00715                   {
00716                       style.Add("font-weight", "bold");
00717                   }
00718                   else
00719                   {
00720                       style.Add("font-weight", "regular");
00721                   }
00722
00723                   if (Font.FontFamily.IsItalic)
00724                   {
00725                       style["font-style"] = "italic";
00726                   }
00727                   else
00728                   {
00729                       style["font-style"] = "normal";
00730                   }
00731
00732                   if (Font.FontFamily.IsOblique)
00733                   {
00734                       style["font-style"] = "oblique";
00735                   }
00736
00737                   string className = GetClass(style);
00738
00739                   textElement.SetAttribute("class", className);
00740               }
00741
00742               if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
00743               {
00744                   ProcessText(text, textElement);
00745               }
00746               else
00747               {
00748                   ProcessGlyphs(text, textElement);
00749               }
00750
00751               if (!string.IsNullOrEmpty(Tag))
00752               {
00753                   textElement.SetAttribute("id", this.TagPrefix + Tag);
00754               }
00755
00756               if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
00757               {
00758                   XmlElement aElement = Document.CreateElement("a", SVGNamespace);
00759                   aElement.SetAttribute("href", destination);
00760                   currentElement.AppendChild(aElement);
00761                   currentElement = aElement;
00762               }
00763
00764               currentElement.AppendChild(textElement);
00765
00766               currentElement = currElement;
00767           }
00768           else
00769           {
00770               PathText(text, x, y);
00771               Fill(FillRule.NonZeroWinding);
00772           }
00773       }
00774
00775       public void LineTo(double x, double y)
00776       {
00777           if (currentFigure.PointCount == 0)
00778           {
00779               currentFigure.StartPoint = new Point(x, y);
00780           }
00781
00782           currentFigure.CurrentPoint = new Point(x, y);
00783           currentFigure.Data += "L " + x.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " " + y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00784           currentFigure.PointCount++;
00785       }
00786
00787       public void MoveTo(double x, double y)
00788       {
00789           if (currentFigure.PointCount > 0)
00790           {
00791               currentPath.Figures.Add(currentFigure);
00792           }
00793
00794           currentFigure = new SVGFigure();
00795           currentFigure.CurrentPoint = new Point(x, y);
00796           currentFigure.StartPoint = new Point(x, y);
00797           currentFigure.Data += "M " + x.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " " + y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00798           currentFigure.PointCount = 1;
```

```
00799         }
00800
00801         public void Rectangle(double x0, double y0, double width, double height)
00802         {
00803             MoveTo(x0, y0);
00804             LineTo(x0 + width, y0);
00805             LineTo(x0 + width, y0 + height);
00806             LineTo(x0, y0 + height);
00807             Close();
00808         }
00809
00810         public void Restore()
00811         {
00812             _transform = states.Pop();
00813             _currClipPath = clipPaths.Pop();
00814             currentPath = new SVGPathObject();
00815             currentFigure = new SVGFigure();
00816         }
00817
00818         public void Rotate(double angle)
00819         {
00820             _transform = MatrixUtils.Rotate(_transform, angle);
00821             currentPath = new SVGPathObject(); currentPath = new SVGPathObject();
00822             currentFigure = new SVGFigure();
00823         }
00824
00825         public void Save()
00826         {
00827             states.Push((double[,])_transform.Clone());
00828             clipPaths.Push(_currClipPath);
00829         }
00830
00831         public void Scale(double scaleX, double scaleY)
00832         {
00833             _transform = MatrixUtils.Scale(_transform, scaleX, scaleY);
00834             currentPath = new SVGPathObject();
00835             currentFigure = new SVGFigure();
00836         }
00837
00838         public void SetFillStyle((int r, int g, int b, double a) style)
00839         {
00840             FillStyle = Colour.FromRgba(style);
00841         }
00842
00843         public void SetFillStyle(Brush style)
00844         {
00845             FillStyle = style;
00846         }
00847
00848         public void SetLineDash(LineDash dash)
00849         {
00850             _lineDash = dash;
00851         }
00852
00853         public void SetStrokeStyle((int r, int g, int b, double a) style)
00854         {
00855             StrokeStyle = Colour.FromRgba(style);
00856         }
00857
00858         public void SetStrokeStyle(Brush style)
00859         {
00860             StrokeStyle = style;
00861         }
00862
00863         public void Stroke()
00864         {
00865             if (currentFigure.PointCount > 0)
00866             {
00867                 currentPath.Figures.Add(currentFigure);
00868             }
00869
00870             XmlElement currElement = currentElement;
00871
00872             if (!string.IsNullOrEmpty(_currClipPath))
00873             {
00874                 currentElement = Document.CreateElement("g", SVGNamespace);
00875                 currentElement.SetAttribute("clip-path", _currClipPath);
00876                 currElement.AppendChild(currentElement);
00877             }
00878
00879             XmlElement path = Document.CreateElement("path", SVGNamespace);
00880             path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
00881
00882             string gradientName = null;
00883
00884             {
00885                 if (StrokeStyle is LinearGradientBrush linearGradient)
```

```
00886                 {
00887                     if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
00888                     {
00889                         if (!string.IsNullOrEmpty(Tag))
00890                         {
00891                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00892                         }
00893                         else
00894                         {
00895                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00896                         }
00897 
00898                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
00899                         this.definitions.AppendChild(gradientElement);
00900 
00901                         if (ReuseGradients)
00902                         {
00903                             gradients.Add(linearGradient, gradientName);
00904                         }
00905 
00906                         gradientCount++;
00907                     }
00908                 }
00909                 else if (StrokeStyle is RadialGradientBrush radialGradient)
00910                 {
00911                     if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
00912                     {
00913                         if (!string.IsNullOrEmpty(Tag))
00914                         {
00915                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00916                         }
00917                         else
00918                         {
00919                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00920                         }
00921 
00922                         XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
00923                         this.definitions.AppendChild(gradientElement);
00924 
00925                         if (ReuseGradients)
00926                         {
00927                             gradients.Add(radialGradient, gradientName);
00928                         }
00929 
00930                         gradientCount++;
00931                     }
00932                 }
00933             }
00934 
00935             if (!UseStyles)
00936             {
00937 
00938                 if (StrokeStyle is SolidColourBrush solid)
00939                 {
00940                     path.SetAttribute("stroke", solid.Colour.ToCSSString(false));
00941                     path.SetAttribute("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00942                 }
00943                 else if (StrokeStyle is LinearGradientBrush linearGradient)
00944                 {
00945                     path.SetAttribute("stroke", "url(#" + gradientName + ")");
00946                 }
00947                 else if (StrokeStyle is RadialGradientBrush radialGradient)
00948                 {
00949                     path.SetAttribute("stroke", "url(#" + gradientName + ")");
00950                 }
00951 
00952                 path.SetAttribute("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
00953 
00954                 switch (LineCap)
00955                 {
00956                     case LineCaps.Butt:
00957                         path.SetAttribute("stroke-linecap", "butt");
00958                         break;
00959                     case LineCaps.Round:
00960                         path.SetAttribute("stroke-linecap", "round");
00961                         break;
00962                     case LineCaps.Square:
00963                         path.SetAttribute("stroke-linecap", "square");
00964                         break;
```

```
00965                 }
00966
00967                 switch (LineJoin)
00968                 {
00969                     case LineJoins.Bevel:
00970                         path.SetAttribute("stroke-linejoin", "bevel");
00971                         break;
00972                     case LineJoins.Round:
00973                         path.SetAttribute("stroke-linejoin", "round");
00974                         break;
00975                     case LineJoins.Miter:
00976                         path.SetAttribute("stroke-linejoin", "miter");
00977                         break;
00978                 }
00979
00980                 if (_lineDash.Phase != 0 || _lineDash.UnitsOn != 0 || _lineDash.UnitsOff != 0)
00981                 {
00982                     path.SetAttribute("stroke-dasharray",
      _lineDash.UnitsOn.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      _lineDash.UnitsOff.ToString(System.Globalization.CultureInfo.InvariantCulture));
00983                     path.SetAttribute("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture));
00984                 }
00985
00986                 path.SetAttribute("fill", "none");
00987                 path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00988                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00989                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00990
00991             }
00992             else
00993             {
00994                 Dictionary<string, string> style = new Dictionary<string, string>();
00995
00996                 if (StrokeStyle is SolidColourBrush solid)
00997                 {
00998                     style.Add("stroke", solid.Colour.ToCSSString(false));
00999                     style.Add("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01000                 }
01001                 else if (StrokeStyle is LinearGradientBrush linearGradient)
01002                 {
01003                     style.Add("stroke", "url(#" + gradientName + ")");
01004                 }
01005                 else if (StrokeStyle is RadialGradientBrush radialGradient)
01006                 {
01007                     style.Add("stroke", "url(#" + gradientName + ")");
01008                 }
01009
01010                 style.Add("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01011
01012                 switch (LineCap)
01013                 {
01014                     case LineCaps.Butt:
01015                         style.Add("stroke-linecap", "butt");
01016                         break;
01017                     case LineCaps.Round:
01018                         style.Add("stroke-linecap", "round");
01019                         break;
01020                     case LineCaps.Square:
01021                         style.Add("stroke-linecap", "square");
01022                         break;
01023                 }
01024
01025                 switch (LineJoin)
01026                 {
01027                     case LineJoins.Bevel:
01028                         style.Add("stroke-linejoin", "bevel");
01029                         break;
01030                     case LineJoins.Round:
01031                         style.Add("stroke-linejoin", "round");
01032                         break;
01033                     case LineJoins.Miter:
01034                         style.Add("stroke-linejoin", "miter");
01035                         break;
01036                 }
01037
01038                 if (_lineDash.Phase != 0 || _lineDash.UnitsOn != 0 || _lineDash.UnitsOff != 0)
01039                 {
01040                     style.Add("stroke-dasharray",
      _lineDash.UnitsOn.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px " +
      _lineDash.UnitsOff.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
```

```
01041                       style.Add("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01042                   }
01043
01044                   style.Add("fill", "none");
01045                   style.Add("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01046                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01047                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01048
01049                   string className = GetClass(style);
01050                   path.SetAttribute("class", className);
01051
01052               }
01053
01054               if (!string.IsNullOrEmpty(Tag))
01055               {
01056                   path.SetAttribute("id", this.TagPrefix + Tag);
01057               }
01058
01059               if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01060               {
01061                   XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01062                   aElement.SetAttribute("href", destination);
01063                   currentElement.AppendChild(aElement);
01064                   currentElement = aElement;
01065               }
01066
01067               currentElement.AppendChild(path);
01068
01069               currentElement = currElement;
01070
01071               currentPath = new SVGPathObject();
01072               currentFigure = new SVGFigure();
01073           }
01074
01075           public void StrokeText(string text, double x, double y)
01076           {
01077               if ((!TextToPaths && TextOption !=
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs) || (TextOption ==
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs && StrokeStyle is SolidColourBrush))
01078               {
01079                   if (!UsedFontFamilies.ContainsKey(Font.FontFamily.FileName))
01080                   {
01081                       UsedFontFamilies.Add(Font.FontFamily.FileName, Font.FontFamily);
01082                       UsedChars.Add(Font.FontFamily.FileName, new HashSet<char>());
01083                   }
01084
01085                   UsedChars[Font.FontFamily.FileName].UnionWith(text);
01086
01087                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
01088
01089                   double[,] currTransform = null;
01090                   double[,] deltaTransform = MatrixUtils.Identity;
01091
01092                   switch (TextBaseline)
01093                   {
01094                       case TextBaselines.Baseline:
01095                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y);
01096                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
01097                           break;
01098                       case TextBaselines.Top:
01099                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Top);
01100                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Top);
01101                           break;
01102                       case TextBaselines.Bottom:
01103                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Bottom);
01104                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Bottom);
01105                           break;
01106                       case TextBaselines.Middle:
01107                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + (metrics.Top + metrics.Bottom) * 0.5);
01108                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + (metrics.Top + metrics.Bottom) * 0.5);
01109                           break;
01110                       default:
01111                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
```

```
      y);
01112                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
01113                         break;
01114                 }
01115
01116                 XmlElement currElement = currentElement;
01117
01118                 if (!string.IsNullOrEmpty(_currClipPath))
01119                 {
01120                     currentElement = Document.CreateElement("g", SVGNamespace);
01121                     currentElement.SetAttribute("clip-path", _currClipPath);
01122                     currElement.AppendChild(currentElement);
01123                 }
01124
01125                 XmlElement textElement;
01126
01127                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01128                 {
01129                     textElement = Document.CreateElement("text", SVGNamespace);
01130                 }
01131                 else
01132                 {
01133                     textElement = Document.CreateElement("g", SVGNamespace);
01134                 }
01135
01136
01137                 string gradientName = null;
01138
01139                 {
01140                     if (StrokeStyle is LinearGradientBrush linearGradient)
01141                     {
01142                         if (!string.IsNullOrEmpty(Tag))
01143                         {
01144                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01145                         }
01146                         else
01147                         {
01148                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01149                         }
01150
01151                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
01152
01153                         deltaTransform = MatrixUtils.Invert(deltaTransform);
01154
01155                         gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01156                         "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01157                         "," + deltaTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01158
01159                         this.definitions.AppendChild(gradientElement);
01160
01161                         gradientCount++;
01162                     }
01163                     else if (StrokeStyle is RadialGradientBrush radialGradient)
01164                     {
01165                         if (!string.IsNullOrEmpty(Tag))
01166                         {
01167                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01168                         }
01169                         else
01170                         {
01171                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01172                         }
01173
01174                         XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
01175
01176                         deltaTransform = MatrixUtils.Invert(deltaTransform);
01177
01178                         gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01179                         "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01180                         "," + deltaTransform[0,
```

```
2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01181
01182                         this.definitions.AppendChild(gradientElement);
01183
01184                         gradientCount++;
01185                     }
01186                 }
01187
01188                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01189                 {
01190                     textElement.SetAttribute("x", "0");
01191                     textElement.SetAttribute("y", "0");
01192                 }
01193
01194                 textElement.SetAttribute("transform", "matrix(" + currTransform[0,
0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01195                         "," + currTransform[0,
1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01196                         "," + currTransform[0,
2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01197
01198                 if (!UseStyles)
01199                 {
01200                     if (StrokeStyle is SolidColourBrush solid)
01201                     {
01202                         textElement.SetAttribute("stroke", solid.Colour.ToCSSString(false));
01203                         textElement.SetAttribute("stroke-opacity",
solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01204                     }
01205                     else if (StrokeStyle is LinearGradientBrush linearGradient)
01206                     {
01207                         textElement.SetAttribute("stroke", "url(#" + gradientName + ")");
01208                     }
01209                     else if (StrokeStyle is RadialGradientBrush radialGradient)
01210                     {
01211                         textElement.SetAttribute("stroke", "url(#" + gradientName + ")");
01212                     }
01213
01214                     if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01215                     {
01216                         textElement.SetAttribute("stroke-width",
LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
01217                     }
01218                     else
01219                     {
01220                         textElement.SetAttribute("stroke-width", (LineWidth *
Font.FontFamily.TrueTypeFile.GetUnitsPerEm() /
Font.FontSize).ToString(System.Globalization.CultureInfo.InvariantCulture));
01221                     }
01222
01223
01224                     switch (LineCap)
01225                     {
01226                         case LineCaps.Butt:
01227                             textElement.SetAttribute("stroke-linecap", "butt");
01228                             break;
01229                         case LineCaps.Round:
01230                             textElement.SetAttribute("stroke-linecap", "round");
01231                             break;
01232                         case LineCaps.Square:
01233                             textElement.SetAttribute("stroke-linecap", "square");
01234                             break;
01235                     }
01236
01237                     switch (LineJoin)
01238                     {
01239                         case LineJoins.Bevel:
01240                             textElement.SetAttribute("stroke-linejoin", "bevel");
01241                             break;
01242                         case LineJoins.Round:
01243                             textElement.SetAttribute("stroke-linejoin", "round");
01244                             break;
01245                         case LineJoins.Miter:
01246                             textElement.SetAttribute("stroke-linejoin", "miter");
01247                             break;
01248                     }
01249
01250                     if (_lineDash.Phase != 0 || _lineDash.UnitsOn != 0 || _lineDash.UnitsOff != 0)
01251                     {
01252                         textElement.SetAttribute("stroke-dasharray",
_lineDash.UnitsOn.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
_lineDash.UnitsOff.ToString(System.Globalization.CultureInfo.InvariantCulture));
01253                         textElement.SetAttribute("stroke-dashoffset",
```

```
    _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture));
01254                   }
01255                   textElement.SetAttribute("fill", "none");
01256
01257                   textElement.SetAttribute("font-size",
    Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture));
01258                   textElement.SetAttribute("font-family", Font.FontFamily.FileName);
01259
01260                   if (Font.FontFamily.IsBold)
01261                   {
01262                       textElement.SetAttribute("font-weight", "bold");
01263                   }
01264                   else
01265                   {
01266                       textElement.SetAttribute("font-weight", "regular");
01267                   }
01268
01269                   if (Font.FontFamily.IsItalic)
01270                   {
01271                       textElement.SetAttribute("font-style", "italic");
01272                   }
01273                   else
01274                   {
01275                       textElement.SetAttribute("font-style", "normal");
01276                   }
01277
01278                   if (Font.FontFamily.IsOblique)
01279                   {
01280                       textElement.SetAttribute("font-style", "oblique");
01281                   }
01282               }
01283               else
01284               {
01285                   Dictionary<string, string> style = new Dictionary<string, string>();
01286
01287                   if (StrokeStyle is SolidColourBrush solid)
01288                   {
01289                       style.Add("stroke", solid.Colour.ToCSSString(false));
01290                       style.Add("stroke-opacity",
    solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01291                   }
01292                   else if (StrokeStyle is LinearGradientBrush linearGradient)
01293                   {
01294                       style.Add("stroke", "url(#" + gradientName + ")");
01295                   }
01296                   else if (StrokeStyle is RadialGradientBrush radialGradient)
01297                   {
01298                       style.Add("stroke", "url(#" + gradientName + ")");
01299                   }
01300
01301                   style.Add("stroke-width",
    LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01302
01303                   switch (LineCap)
01304                   {
01305                       case LineCaps.Butt:
01306                           style.Add("stroke-linecap", "butt");
01307                           break;
01308                       case LineCaps.Round:
01309                           style.Add("stroke-linecap", "round");
01310                           break;
01311                       case LineCaps.Square:
01312                           style.Add("stroke-linecap", "square");
01313                           break;
01314                   }
01315
01316                   switch (LineJoin)
01317                   {
01318                       case LineJoins.Bevel:
01319                           style.Add("stroke-linejoin", "bevel");
01320                           break;
01321                       case LineJoins.Round:
01322                           style.Add("stroke-linejoin", "round");
01323                           break;
01324                       case LineJoins.Miter:
01325                           style.Add("stroke-linejoin", "miter");
01326                           break;
01327                   }
01328
01329                   if (_lineDash.Phase != 0 || _lineDash.UnitsOn != 0 || _lineDash.UnitsOff != 0)
01330                   {
01331                       style.Add("stroke-dasharray",
    _lineDash.UnitsOn.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px " +
    _lineDash.UnitsOff.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01332                       style.Add("stroke-dashoffset",
    _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01333                   }
```

```
01334                     style.Add("fill", "none");
01335
01336                     style.Add("font-size",
      Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01337                     style.Add("font-family", Font.FontFamily.FileName);
01338
01339                     if (Font.FontFamily.IsBold)
01340                     {
01341                         style.Add("font-weight", "bold");
01342                     }
01343                     else
01344                     {
01345                         style.Add("font-weight", "regular");
01346                     }
01347
01348                     if (Font.FontFamily.IsItalic)
01349                     {
01350                         style["font-style"] = "italic";
01351                     }
01352                     else
01353                     {
01354                         style["font-style"] = "normal";
01355                     }
01356
01357                     if (Font.FontFamily.IsOblique)
01358                     {
01359                         style["font-style"] = "oblique";
01360                     }
01361
01362                     string className = GetClass(style);
01363
01364                     textElement.SetAttribute("class", className);
01365                 }
01366
01367                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01368                 {
01369                     ProcessText(text, textElement);
01370                 }
01371                 else
01372                 {
01373                     ProcessGlyphs(text, textElement);
01374                 }
01375
01376                 if (!string.IsNullOrEmpty(Tag))
01377                 {
01378                     textElement.SetAttribute("id", this.TagPrefix + Tag);
01379                 }
01380
01381                 if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01382                 {
01383                     XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01384                     aElement.SetAttribute("href", destination);
01385                     currentElement.AppendChild(aElement);
01386                     currentElement = aElement;
01387                 }
01388
01389                 currentElement.AppendChild(textElement);
01390
01391                 currentElement = currElement;
01392             }
01393             else
01394             {
01395                 PathText(text, x, y);
01396                 Stroke();
01397             }
01398         }
01399
01400         public void Transform(double a, double b, double c, double d, double e, double f)
01401         {
01402             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
01403             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
01404
01405             currentPath = new SVGPathObject();
01406             currentFigure = new SVGFigure();
01407         }
01408
01409         public void Translate(double x, double y)
01410         {
01411             _transform = MatrixUtils.Translate(_transform, x, y);
01412
01413             currentPath = new SVGPathObject();
01414             currentFigure = new SVGFigure();
01415         }
01416
01417         public void SetClippingPath()
01418         {
```

```
01419             if (currentFigure.PointCount > 0)
01420             {
01421                 currentPath.Figures.Add(currentFigure);
01422             }
01423 
01424             XmlElement clipPath = Document.CreateElement("clipPath", SVGNamespace);
01425             string id = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag) :
      Guid.NewGuid().ToString("N");
01426             clipPath.SetAttribute("id", id);
01427 
01428             if (!string.IsNullOrEmpty(_currClipPath))
01429             {
01430                 clipPath.SetAttribute("clip-path", _currClipPath);
01431             }
01432 
01433             XmlElement path = Document.CreateElement("path", SVGNamespace);
01434             path.SetAttribute("id", id + "_clipPath");
01435             path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
01436 
01437             if (StrokeStyle is SolidColourBrush solid)
01438             {
01439                 path.SetAttribute("stroke", solid.Colour.ToCSSString(false));
01440                 path.SetAttribute("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01441             }
01442             else if (StrokeStyle is LinearGradientBrush linearGradient)
01443             {
01444                 string gradientName;
01445 
01446                 if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
01447                 {
01448                     if (!string.IsNullOrEmpty(Tag))
01449                     {
01450                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01451                     }
01452                     else
01453                     {
01454                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01455                     }
01456 
01457                     XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
01458                     this.definitions.AppendChild(gradientElement);
01459 
01460                     if (ReuseGradients)
01461                     {
01462                         gradients.Add(linearGradient, gradientName);
01463                     }
01464 
01465                     gradientCount++;
01466                 }
01467 
01468                 path.SetAttribute("stroke", "url(#" + gradientName + ")");
01469             }
01470             else if (StrokeStyle is RadialGradientBrush radialGradient)
01471             {
01472                 string gradientName;
01473 
01474                 if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
01475                 {
01476                     if (!string.IsNullOrEmpty(Tag))
01477                     {
01478                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01479                     }
01480                     else
01481                     {
01482                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01483                     }
01484 
01485                     XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
01486                     this.definitions.AppendChild(gradientElement);
01487 
01488                     if (ReuseGradients)
01489                     {
01490                         gradients.Add(radialGradient, gradientName);
01491                     }
01492 
01493                     gradientCount++;
01494                 }
01495 
01496                 path.SetAttribute("stroke", "url(#" + gradientName + ")");
01497             }
```

```
01498
01499             path.SetAttribute("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
01500
01501             switch (LineCap)
01502             {
01503                 case LineCaps.Butt:
01504                     path.SetAttribute("stroke-linecap", "butt");
01505                     break;
01506                 case LineCaps.Round:
01507                     path.SetAttribute("stroke-linecap", "round");
01508                     break;
01509                 case LineCaps.Square:
01510                     path.SetAttribute("stroke-linecap", "square");
01511                     break;
01512             }
01513
01514             switch (LineJoin)
01515             {
01516                 case LineJoins.Bevel:
01517                     path.SetAttribute("stroke-linejoin", "bevel");
01518                     break;
01519                 case LineJoins.Round:
01520                     path.SetAttribute("stroke-linejoin", "round");
01521                     break;
01522                 case LineJoins.Miter:
01523                     path.SetAttribute("stroke-linejoin", "miter");
01524                     break;
01525             }
01526
01527             if (_lineDash.Phase != 0 || _lineDash.UnitsOn != 0 || _lineDash.UnitsOff != 0)
01528             {
01529                 path.SetAttribute("stroke-dasharray",
      _lineDash.UnitsOn.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      _lineDash.UnitsOff.ToString(System.Globalization.CultureInfo.InvariantCulture));
01530                 path.SetAttribute("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture));
01531             }
01532
01533             path.SetAttribute("fill", "none");
01534             path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01535                 "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01536                 "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01537
01538             clipPath.AppendChild(path);
01539
01540             currentElement.AppendChild(clipPath);
01541
01542             _currClipPath = "url(#" + id + ")";
01543
01544             currentPath = new SVGPathObject();
01545             currentFigure = new SVGFigure();
01546         }
01547
01548         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
01549         {
01550             Save();
01551
01552             MoveTo(destinationX, destinationY);
01553             LineTo(destinationX + destinationWidth, destinationY);
01554             LineTo(destinationX + destinationWidth, destinationY + destinationHeight);
01555             LineTo(destinationX, destinationY + destinationHeight);
01556             Close();
01557             SetClippingPath();
01558
01559             double sourceRectX = (double)sourceX / image.Width;
01560             double sourceRectY = (double)sourceY / image.Height;
01561             double sourceRectWidth = (double)sourceWidth / image.Width;
01562             double sourceRectHeight = (double)sourceHeight / image.Height;
01563
01564             double scaleX = destinationWidth / sourceRectWidth;
01565             double scaleY = destinationHeight / sourceRectHeight;
01566
01567             double translationX = destinationX / scaleX - sourceRectX;
01568             double translationY = destinationY / scaleY - sourceRectY;
01569
01570             Scale(scaleX, scaleY);
01571             Translate(translationX, translationY);
01572
01573             XmlElement currElement = currentElement;
01574
```

```
01575             if (!string.IsNullOrEmpty(_currClipPath))
01576             {
01577                 currentElement = Document.CreateElement("g", SVGNamespace);
01578                 currentElement.SetAttribute("clip-path", _currClipPath);
01579                 currElement.AppendChild(currentElement);
01580             }
01581 
01582             XmlElement img = Document.CreateElement("image", SVGNamespace);
01583             img.SetAttribute("x", "0");
01584             img.SetAttribute("y", "0");
01585 
01586             img.SetAttribute("width", "1");
01587             img.SetAttribute("height", "1");
01588 
01589             img.SetAttribute("preserveAspectRatio", "none");
01590 
01591             if (image.Interpolate)
01592             {
01593                 img.SetAttribute("image-rendering", "optimizeQuality");
01594             }
01595             else
01596             {
01597                 img.SetAttribute("image-rendering", "pixelated");
01598             }
01599 
01600             img.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01601                 "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01602                 "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01603 
01604             if (!string.IsNullOrEmpty(Tag))
01605             {
01606                 img.SetAttribute("id", this.TagPrefix + Tag);
01607             }
01608 
01609             img.SetAttribute("href", "http://www.w3.org/1999/xlink", "" +
      Convert.ToBase64String(image.PNGStream.ToArray()));
01610 
01611             if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01612             {
01613                 XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01614                 aElement.SetAttribute("href", destination);
01615                 currentElement.AppendChild(aElement);
01616                 currentElement = aElement;
01617             }
01618 
01619             currentElement.AppendChild(img);
01620 
01621             currentElement = currElement;
01622 
01623             Restore();
01624         }
01625 
01626         private void ProcessText(string text, XmlNode parent)
01627         {
01628             if (Font.EnableKerning && this.TextOption ==
      SVGContextInterpreter.TextOptions.SubsetFonts)
01629             {
01630                 List<(string, Point)> tSpans = new List<(string, Point)>();
01631 
01632                 StringBuilder currentRun = new StringBuilder();
01633                 Point currentKerning = new Point();
01634 
01635                 Point currentGlyphPlacementDelta = new Point();
01636                 Point currentGlyphAdvanceDelta = new Point();
01637                 Point nextGlyphPlacementDelta = new Point();
01638                 Point nextGlyphAdvanceDelta = new Point();
01639 
01640                 for (int i = 0; i < text.Length; i++)
01641                 {
01642                     if (i < text.Length - 1)
01643                     {
01644                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01645                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01646                         nextGlyphAdvanceDelta = new Point();
01647                         nextGlyphPlacementDelta = new Point();
01648 
01649                         TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01650 
01651                         if (kerning != null)
01652                         {
01653                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
```

```
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01654                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01655
01656                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01657                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01658                         }
01659                     }
01660
01661                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
01662                     {
01663                         if (currentRun.Length > 0)
01664                         {
01665                             tSpans.Add((currentRun.ToString(), currentKerning));
01666
01667                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
01668
01669                             currentRun.Clear();
01670                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01671                         }
01672                         else
01673                         {
01674                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      currentKerning.Y)));
01675
01676                             currentRun.Clear();
01677                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01678                         }
01679                     }
01680                     else
01681                     {
01682                         currentRun.Append(text[i]);
01683                     }
01684                 }
01685
01686                 if (currentRun.Length > 0)
01687                 {
01688                     tSpans.Add((currentRun.ToString(), currentKerning));
01689                 }
01690
01691                 for (int i = 0; i < tSpans.Count; i++)
01692                 {
01693                     XmlElement tspanElement = Document.CreateElement("tspan", SVGNamespace);
01694                     tspanElement.InnerText = tSpans[i].Item1.Replace(" ", "\u00A0");
01695
01696                     if (tSpans[i].Item2.X != 0)
01697                     {
01698                         tspanElement.SetAttribute("dx",
      tSpans[i].Item2.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01699                     }
01700
01701                     if (tSpans[i].Item2.Y != 0)
01702                     {
01703                         tspanElement.SetAttribute("dy",
      tSpans[i].Item2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01704                     }
01705
01706                     parent.AppendChild(tspanElement);
01707                 }
01708
01709             }
01710             else
01711             {
01712                 parent.InnerText = text.Replace(" ", "\u00A0");
01713             }
01714         }
01715
01716         private void ProcessGlyphs(string text, XmlNode parent)
01717         {
01718             List<(string, Point)> tSpans = new List<(string, Point)>();
01719
01720             Point currentGlyphPlacementDelta = new Point();
01721             Point currentGlyphAdvanceDelta = new Point();
01722             Point nextGlyphPlacementDelta = new Point();
01723             Point nextGlyphAdvanceDelta = new Point();
01724
01725             double currX = 0;
01726             double currY = 0;
```

```
01727
01728             for (int i = 0; i < text.Length; i++)
01729             {
01730                 if (i < text.Length - 1)
01731                 {
01732                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01733                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01734                     nextGlyphAdvanceDelta = new Point();
01735                     nextGlyphPlacementDelta = new Point();
01736
01737                     TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01738
01739                     if (kerning != null)
01740                     {
01741                         currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01742                         currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01743
01744                         nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01745                         nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01746                     }
01747                 }
01748
01749                 tSpans.Add((text[i].ToString(), new Point(currX, currY)));
01750
01751                 double advanceWidth = Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) *
      Font.FontSize / 1000;
01752                 currX += advanceWidth + currentGlyphPlacementDelta.X * Font.FontSize / 1000 +
      (currentGlyphAdvanceDelta.X - currentGlyphPlacementDelta.X) * Font.FontSize / 1000;
01753                 currY += currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      (currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y) * Font.FontSize / 1000;
01754             }
01755
01756             double scale = Font.FontSize / Font.FontFamily.TrueTypeFile.GetUnitsPerEm();
01757
01758             string scaleText = scale.ToString(System.Globalization.CultureInfo.InvariantCulture);
01759
01760             ((XmlElement)parent).SetAttribute("transform",
      ((XmlElement)parent).GetAttribute("transform") + ", scale(" + scaleText + ")");
01761
01762             for (int i = 0; i < tSpans.Count; i++)
01763             {
01764                 XmlElement useElement = Document.CreateElement("use", SVGNamespace);
01765                 useElement.SetAttribute("href", "#" + Font.FontFamily.FileName + "-" +
      tSpans[i].Item1);
01766
01767                 useElement.SetAttribute("transform", "translate(" + (tSpans[i].Item2.X /
      scale).ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (tSpans[i].Item2.Y /
      scale).ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01768                 parent.AppendChild(useElement);
01769             }
01770         }
01771
01772         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
01773         {
01774             if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.IgnoreAll)
01775             {
01776                 graphics.CopyToIGraphicsContext(this);
01777             }
01778             else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.SkipAll)
01779             {
01780
01781             }
01782             else
01783             {
01784                 bool rasterisationNeeded = false;
01785                 bool justDraw = false;
01786
01787                 if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.RasteriseAll)
01788                 {
01789                     rasterisationNeeded = true;
01790                     justDraw = false;
01791                 }
01792                 else
01793                 {
01794                     if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.RasteriseIfNecessary)
01795                     {
01796                         justDraw = false;
01797                         rasterisationNeeded = true;
```

```
01798                     }
01799                     else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.NeverRasteriseAndIgnore)
01800                     {
01801                         rasterisationNeeded = false;
01802                         justDraw = true;
01803                     }
01804                     else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.NeverRasteriseAndSkip)
01805                     {
01806                         rasterisationNeeded = false;
01807                         justDraw = false;
01808                     }
01809
01810                     if (filter is MaskFilter mask)
01811                     {
01812                         rasterisationNeeded = false;
01813                         justDraw = false;
01814
01815                         XmlElement currElement = currentElement;
01816
01817                         if (!string.IsNullOrEmpty(_currClipPath))
01818                         {
01819                             currentElement = Document.CreateElement("g", SVGNamespace);
01820                             currentElement.SetAttribute("clip-path", _currClipPath);
01821                             currElement.AppendChild(currentElement);
01822                         }
01823
01824                         XmlElement currentElement2 = currentElement;
01825
01826                         string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
      "@filter") :  Guid.NewGuid().ToString("N");
01827
01828                         Rectangle bounds = graphics.GetBounds();
01829
01830                         Point p1 = new Point(bounds.Location.X, bounds.Location.Y);
01831                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width,
      bounds.Location.Y);
01832                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width, bounds.Location.Y
      + bounds.Size.Height);
01833                         Point p4 = new Point(bounds.Location.X, bounds.Location.Y +
      bounds.Size.Height);
01834
01835                         /*p1 = MatrixUtils.Multiply(_transform, p1);
01836 p2 = MatrixUtils.Multiply(_transform, p2);
01837 p3 = MatrixUtils.Multiply(_transform, p3);
01838 p4 = MatrixUtils.Multiply(_transform, p4);*/
01839
01840                         bounds = Point.Bounds(p1, p2, p3, p4);
01841
01842                         XmlElement maskElement = Document.CreateElement("mask", SVGNamespace);
01843                         maskElement.SetAttribute("id", filterGuid);
01844                         maskElement.SetAttribute("maskUnits", "userSpaceOnUse");
01845                         maskElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01846                         maskElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01847                         maskElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
01848                         maskElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
01849
01850                         this.definitions.AppendChild(maskElement);
01851
01852                         currentElement = maskElement;
01853
01854                         double[,] currTransform = _transform;
01855                         _transform = MatrixUtils.Identity;
01856
01857                         mask.Mask.CopyToIGraphicsContext(this);
01858
01859                         _transform = currTransform;
01860
01861                         currentElement = Document.CreateElement("g", SVGNamespace);
01862
01863                         if (!string.IsNullOrEmpty(this.Tag))
01864                         {
01865                             currentElement.SetAttribute("id", this.TagPrefix + Tag);
01866                         }
01867
01868                         currentElement.SetAttribute("mask", "url(#" + filterGuid + ")");
01869                         currentElement2.AppendChild(currentElement);
01870
01871                         currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01872                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
```

```
    + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01873                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
    + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01874                         currTransform = _transform;
01875                         _transform = MatrixUtils.Identity;
01876
01877                         graphics.CopyToIGraphicsContext(this);
01878
01879                         _transform = currTransform;
01880
01881                         currentElement = currElement;
01882                     }
01883                     if (filter is GaussianBlurFilter gauss)
01884                     {
01885                         rasterisationNeeded = false;
01886                         justDraw = false;
01887                         XmlElement currElement = currentElement;
01888
01889                         if (!string.IsNullOrEmpty(_currClipPath))
01890                         {
01891                             currentElement = Document.CreateElement("g", SVGNamespace);
01892                             currentElement.SetAttribute("clip-path", _currClipPath);
01893                             currElement.AppendChild(currentElement);
01894                         }
01895
01896                         Rectangle bounds = graphics.GetBounds();
01897
01898                         Point p1 = new Point(bounds.Location.X - gauss.StandardDeviation * 3,
    bounds.Location.Y - gauss.StandardDeviation * 3);
01899                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width +
    gauss.StandardDeviation * 3, bounds.Location.Y - gauss.StandardDeviation * 3);
01900                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width +
    gauss.StandardDeviation * 3, bounds.Location.Y + bounds.Size.Height + gauss.StandardDeviation * 3);
01901                         Point p4 = new Point(bounds.Location.X - gauss.StandardDeviation * 3,
    bounds.Location.Y + bounds.Size.Height + gauss.StandardDeviation * 3);
01902
01903                         /*p1 = MatrixUtils.Multiply(_transform, p1);
01904 p2 = MatrixUtils.Multiply(_transform, p2);
01905 p3 = MatrixUtils.Multiply(_transform, p3);
01906 p4 = MatrixUtils.Multiply(_transform, p4);*/
01907
01908                         bounds = Point.Bounds(p1, p2, p3, p4);
01909
01910                         string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
    "@filter") :  Guid.NewGuid().ToString("N");
01911
01912                         XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
01913                         filterElement.SetAttribute("id", filterGuid);
01914                         filterElement.SetAttribute("color-interpolation-filters", "sRGB");
01915                         filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
01916                         filterElement.SetAttribute("x",
    bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01917                         filterElement.SetAttribute("y",
    bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01918                         filterElement.SetAttribute("width",
    bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
01919                         filterElement.SetAttribute("height",
    bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
01920
01921                         this.definitions.AppendChild(filterElement);
01922
01923                         XmlElement feElement = Document.CreateElement("feGaussianBlur", SVGNamespace);
01924                         feElement.SetAttribute("stdDeviation",
    gauss.StandardDeviation.ToString(System.Globalization.CultureInfo.InvariantCulture));
01925
01926                         if (!string.IsNullOrEmpty(Tag))
01927                         {
01928                             feElement.SetAttribute("id", this.TagPrefix + Tag + "@feGaussianBlur");
01929                         }
01930
01931                         filterElement.AppendChild(feElement);
01932
01933                         XmlElement currentElement2 = currentElement;
01934
01935                         currentElement = Document.CreateElement("g", SVGNamespace);
01936
01937                         if (!string.IsNullOrEmpty(this.Tag))
01938                         {
01939                             currentElement.SetAttribute("id", this.TagPrefix + Tag);
01940                         }
01941
01942                         currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
01943                         currentElement2.AppendChild(currentElement);
01944
01945                         currentElement.SetAttribute("transform", "matrix(" + _transform[0,
    0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
    0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
```

```
01946                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01947                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01948                         double[,] currTransform = _transform;
01949                         _transform = MatrixUtils.Identity;
01950 
01951                         graphics.CopyToIGraphicsContext(this);
01952 
01953                         currentElement = currElement;
01954                     }
01955                     else if (filter is ColourMatrixFilter cmf)
01956                     {
01957                         rasterisationNeeded = false;
01958                         justDraw = false;
01959                         XmlElement currElement = currentElement;
01960 
01961                         if (!string.IsNullOrEmpty(_currClipPath))
01962                         {
01963                             currentElement = Document.CreateElement("g", SVGNamespace);
01964                             currentElement.SetAttribute("clip-path", _currClipPath);
01965                             currElement.AppendChild(currentElement);
01966                         }
01967 
01968                         Rectangle bounds = graphics.GetBounds();
01969 
01970                         Point p1 = new Point(bounds.Location.X, bounds.Location.Y);
01971                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width,
      bounds.Location.Y);
01972                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width, bounds.Location.Y
      + bounds.Size.Height);
01973                         Point p4 = new Point(bounds.Location.X, bounds.Location.Y +
      bounds.Size.Height);
01974 
01975                         /* p1 = MatrixUtils.Multiply(_transform, p1);
01976 p2 = MatrixUtils.Multiply(_transform, p2);
01977 p3 = MatrixUtils.Multiply(_transform, p3);
01978 p4 = MatrixUtils.Multiply(_transform, p4);*/
01979 
01980                         bounds = Point.Bounds(p1, p2, p3, p4);
01981 
01982                         string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
      "@filter") :  Guid.NewGuid().ToString("N");
01983 
01984                         XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
01985                         filterElement.SetAttribute("id", filterGuid);
01986                         filterElement.SetAttribute("color-interpolation-filters", "sRGB");
01987                         filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
01988                         filterElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01989                         filterElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01990                         filterElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
01991                         filterElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
01992 
01993                         this.definitions.AppendChild(filterElement);
01994 
01995                         XmlElement feElement = Document.CreateElement("feColorMatrix", SVGNamespace);
01996                         feElement.SetAttribute("type", "matrix");
01997 
01998                         if (!string.IsNullOrEmpty(Tag))
01999                         {
02000                             feElement.SetAttribute("id", this.TagPrefix + Tag + "@feColorMatrix");
02001                         }
02002 
02003                         StringBuilder matrix = new StringBuilder();
02004 
02005                         for (int i = 0; i < 4; i++)
02006                         {
02007                             for (int j = 0; j < 5; j++)
02008                             {
02009                                 matrix.Append(cmf.ColourMatrix[i,
      j].ToString(System.Globalization.CultureInfo.InvariantCulture));
02010                                 if (i != 3 || j != 4)
02011                                 {
02012                                     matrix.Append(" ");
02013                                 }
02014                             }
02015                         }
02016 
02017                         feElement.SetAttribute("values", matrix.ToString());
02018                         filterElement.AppendChild(feElement);
02019 
02020                         XmlElement currentElement2 = currentElement;
02021 
```

```
02022                         currentElement = Document.CreateElement("g", SVGNamespace);
02023
02024                         if (!string.IsNullOrEmpty(this.Tag))
02025                         {
02026                             currentElement.SetAttribute("id", this.TagPrefix + Tag);
02027                         }
02028
02029                         currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
02030                         currentElement2.AppendChild(currentElement);
02031
02032                         currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02033                         "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02034                         "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
02035                         double[,] currTransform = _transform;
02036                         _transform = MatrixUtils.Identity;
02037
02038                         graphics.CopyToIGraphicsContext(this);
02039
02040                         currentElement = currElement;
02041                     }
02042                     else if (filter is CompositeLocationInvariantFilter comp)
02043                     {
02044                         bool allSupported = true;
02045
02046                         foreach (IFilter filter2 in comp.Filters)
02047                         {
02048                             if (!(filter2 is GaussianBlurFilter) && !(filter2 is ColourMatrixFilter))
02049                             {
02050                                 allSupported = false;
02051                                 break;
02052                             }
02053                         }
02054
02055                         if (allSupported)
02056                         {
02057                             rasterisationNeeded = false;
02058                             justDraw = false;
02059
02060                             XmlElement currElement = currentElement;
02061
02062                             if (!string.IsNullOrEmpty(_currClipPath))
02063                             {
02064                                 currentElement = Document.CreateElement("g", SVGNamespace);
02065                                 currentElement.SetAttribute("clip-path", _currClipPath);
02066                                 currElement.AppendChild(currentElement);
02067                             }
02068
02069                             Rectangle bounds = graphics.GetBounds();
02070
02071                             Point p1 = new Point(bounds.Location.X - comp.TopLeftMargin.X,
      bounds.Location.Y - comp.TopLeftMargin.Y);
02072                             Point p2 = new Point(bounds.Location.X + bounds.Size.Width +
      comp.BottomRightMargin.X, bounds.Location.Y - comp.TopLeftMargin.Y);
02073                             Point p3 = new Point(bounds.Location.X + bounds.Size.Width +
      comp.BottomRightMargin.X, bounds.Location.Y + bounds.Size.Height + comp.BottomRightMargin.Y);
02074                             Point p4 = new Point(bounds.Location.X - comp.TopLeftMargin.X,
      bounds.Location.Y + bounds.Size.Height + comp.BottomRightMargin.Y);
02075
02076                             /* p1 = MatrixUtils.Multiply(_transform, p1);
02077 p2 = MatrixUtils.Multiply(_transform, p2);
02078 p3 = MatrixUtils.Multiply(_transform, p3);
02079 p4 = MatrixUtils.Multiply(_transform, p4);*/
02080
02081                             bounds = Point.Bounds(p1, p2, p3, p4);
02082
02083                             string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
      "@filter") :  Guid.NewGuid().ToString("N");
02084
02085                             XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
02086                             filterElement.SetAttribute("id", filterGuid);
02087                             filterElement.SetAttribute("color-interpolation-filters", "sRGB");
02088                             filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
02089                             filterElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
02090                             filterElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
02091                             filterElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
02092                             filterElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02093
02094                             this.definitions.AppendChild(filterElement);
02095
```

```
02096                               int index = 0;
02097
02098                               foreach (IFilter filter2 in comp.Filters)
02099                               {
02100                                   if (filter2 is GaussianBlurFilter gauss2)
02101                                   {
02102                                       XmlElement feElement = Document.CreateElement("feGaussianBlur",
      SVGNamespace);
02103                                       feElement.SetAttribute("stdDeviation",
      gauss2.StandardDeviation.ToString(System.Globalization.CultureInfo.InvariantCulture));
02104
02105                                       if (!string.IsNullOrEmpty(Tag))
02106                                       {
02107                                           filterElement.SetAttribute("id", this.TagPrefix + Tag +
      "@feGaussianBlur" + index.ToString());
02108                                       }
02109
02110                                       filterElement.AppendChild(feElement);
02111                                   }
02112                                   else if (filter2 is ColourMatrixFilter cmf2)
02113                                   {
02114                                       XmlElement feElement = Document.CreateElement("feColorMatrix",
      SVGNamespace);
02115                                       feElement.SetAttribute("type", "matrix");
02116                                       if (!string.IsNullOrEmpty(Tag))
02117                                       {
02118                                           filterElement.SetAttribute("id", this.TagPrefix + Tag +
      "@feColorMatrix" + index.ToString());
02119                                       }
02120
02121                                       StringBuilder matrix = new StringBuilder();
02122
02123                                       for (int i = 0; i < 4; i++)
02124                                       {
02125                                           for (int j = 0; j < 5; j++)
02126                                           {
02127                                               matrix.Append(cmf2.ColourMatrix[i,
      j].ToString(System.Globalization.CultureInfo.InvariantCulture));
02128                                               if (i != 3 || j != 4)
02129                                               {
02130                                                   matrix.Append(" ");
02131                                               }
02132                                           }
02133                                       }
02134
02135                                       feElement.SetAttribute("values", matrix.ToString());
02136                                       filterElement.AppendChild(feElement);
02137                                   }
02138
02139                                   index++;
02140                               }
02141
02142                               XmlElement currentElement2 = currentElement;
02143
02144                               currentElement = Document.CreateElement("g", SVGNamespace);
02145
02146                               if (!string.IsNullOrEmpty(this.Tag))
02147                               {
02148                                   currentElement.SetAttribute("id", this.TagPrefix + Tag);
02149                               }
02150
02151                               currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
02152                               currentElement2.AppendChild(currentElement);
02153
02154                               currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02155                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02156                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
02157                               double[,] currTransform = _transform;
02158                               _transform = MatrixUtils.Identity;
02159
02160                               graphics.CopyToIGraphicsContext(this);
02161
02162                               currentElement = currElement;
02163
02164
02165                           }
02166                       }
02167                   }
02168
02169                   if (rasterisationNeeded)
02170                   {
02171                       double scale = FilterOption.RasterisationResolution;
02172
```

```
02173                     Rectangle bounds = graphics.GetBounds();
02174
02175                     bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X,
      bounds.Location.Y - filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X +
      filter.BottomRightMargin.X, bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
02176
02177                     if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02178                     {
02179                         if (!FilterOption.RasterisationResolutionRelative)
02180                         {
02181                             scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02182                         }
02183
02184                         if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
02185                         {
02186                             RasterImage filtered = null;
02187
02188                             if (filter is ILocationInvariantFilter locInvFilter)
02189                             {
02190                                 filtered = locInvFilter.Filter(rasterised, scale);
02191                             }
02192                             else if (filter is IFilterWithLocation filterWithLoc)
02193                             {
02194                                 filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02195                             }
02196
02197                             if (filtered != null)
02198                             {
02199                                 rasterised.Dispose();
02200
02201                                 DrawRasterImage(0, 0, filtered.Width, filtered.Height,
      bounds.Location.X, bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02202                             }
02203                         }
02204                         else
02205                         {
02206                             throw new NotImplementedException(@"The filter could not be rasterised!
      You can avoid this error by doing one of the following:
02207 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
02208 • Provide your own implementation of Graphics.RasterisationMethod.
02209 • Set the FilterOption.Operation to ""NeverRasteriseAndIgnore"", ""NeverRasteriseAndSkip"",
      ""IgnoreAll"" or ""SkipAll"".");
02210                         }
02211                     }
02212                 }
02213
02214                 if (justDraw)
02215                 {
02216                     graphics.CopyToIGraphicsContext(this);
02217                 }
02218             }
02219         }
02220     }
02221
02222
02223 /// <summary>
02224 /// Contains methods to render a <see cref="Page"/> as an SVG file.
02225 /// </summary>
02226     public static class SVGContextInterpreter
02227     {
02228
02229 /// <summary>
02230 /// Render the page to an SVG file.
02231 /// </summary>
02232 /// <param name="page">The <see cref="Page"/> to render.</param>
02233 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
02234 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02235 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02236 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02237 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
      attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
      presentation attributes.</param>
02238         public static void SaveAsSVG(this Page page, string fileName, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default, bool useStyles = false)
02239         {
02240             using (FileStream sr = new FileStream(fileName, FileMode.Create))
02241             {
02242                 page.SaveAsSVG(sr, textOption, linkDestinations, filterOption, useStyles);
02243             }
```

```
02244         }
02245
02246 /// <summary>
02247 /// Defines whether the used fonts should be included in the file.
02248 /// </summary>
02249         public enum TextOptions
02250         {
02251 /// <summary>
02252 /// Embeds the full font files.
02253 /// </summary>
02254             EmbedFonts,
02255
02256 /// <summary>
02257 /// Embeds subsetted font files containing only the glyphs for the characters that have been used.
02258 /// </summary>
02259             SubsetFonts,
02260
02261 /// <summary>
02262 /// Does not embed any font file and converts all text items into paths.
02263 /// </summary>
02264             ConvertIntoPaths,
02265
02266 /// <summary>
02267 /// Does not embed any font file, but still encodes text items as such.
02268 /// </summary>
02269             DoNotEmbed,
02270
02271 /// <summary>
02272 /// Converts all text items into paths, but defines each glyph only once and reuses it when needed.
02273 /// </summary>
02274             ConvertIntoPathsUsingGlyphs
02275         }
02276
02277 /// <summary>
02278 /// Determines how and whether image filters are rasterised.
02279 /// </summary>
02280         public class FilterOption
02281         {
02282 /// <summary>
02283 /// Defines whether image filters should be rasterised or not.
02284 /// </summary>
02285             public enum FilterOperations
02286             {
02287 /// <summary>
02288 /// Image filters will always be rasterised.
02289 /// </summary>
02290                 RasteriseAll,
02291
02292 /// <summary>
02293 /// Image filters will only be rasterised if they are not supported natively by the output file
      format.
02294 /// </summary>
02295                 RasteriseIfNecessary,
02296
02297 /// <summary>
02298 /// Image filters will never be rasterised; for filters that are not supported, the filter will be
      ignored.
02299 /// </summary>
02300                 NeverRasteriseAndIgnore,
02301
02302 /// <summary>
02303 /// Image filters will never be rasterised; if an image should be drawn with an unsupported filter,
      the image will not be drawn at all.
02304 /// </summary>
02305                 NeverRasteriseAndSkip,
02306
02307 /// <summary>
02308 /// All image filters (supported and unsupported) will be ignored.
02309 /// </summary>
02310                 IgnoreAll,
02311
02312 /// <summary>
02313 /// All the images that should be drawn with a filter will be ignored.
02314 /// </summary>
02315                 SkipAll
02316             }
02317
02318 /// <summary>
02319 /// Defines whether image filters should be rasterised or not.
02320 /// </summary>
02321             public FilterOperations Operation { get; } = FilterOperations.RasteriseIfNecessary;
02322
02323 /// <summary>
02324 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
02325 /// </summary>
```

```
02326             public double RasterisationResolution { get; } = 1;
02327
02328 /// <summary>
02329 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
02330 /// </summary>
02331             public bool RasterisationResolutionRelative { get; } = true;
02332
02333 /// <summary>
02334 /// The default options for image filter rasterisation.
02335 /// </summary>
02336             public static FilterOption Default = new
      FilterOption(FilterOperations.RasteriseIfNecessary, 1, true);
02337
02338 /// <summary>
02339 /// Create a new <see cref="FilterOption"/> object.
02340 /// </summary>
02341 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
02342 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
02343 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
02344             public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
02345             {
02346                 this.Operation = operation;
02347                 this.RasterisationResolution = rasterisationResolution;
02348                 this.RasterisationResolutionRelative = rasterisationResolutionRelative;
02349             }
02350         }
02351
02352 /// <summary>
02353 /// Render the page to an SVG stream.
02354 /// </summary>
02355 /// <param name="page">The <see cref="Page"/> to render.</param>
02356 /// <param name="stream">The stream to which the SVG data will be written.</param>
02357 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02358 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02359 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02360 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
      attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
      presentation attributes.</param>
02361         public static void SaveAsSVG(this Page page, Stream stream, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default, bool useStyles = false)
02362         {
02363             XmlDocument doc = page.SaveAsSVG(textOption, linkDestinations, filterOption, useStyles);
02364             WriteXMLToStream(doc.DocumentElement, stream);
02365         }
02366
02367 /// <summary>
02368 /// Render the page to an SVG document.
02369 /// </summary>
02370 /// <param name="page">The <see cref="Page"/> to render.</param>
02371 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02372 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02373 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02374 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
      attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
      presentation attributes.</param>
02375 /// <returns>An <see cref="XmlDocument"/> containing the rendered SVG image.</returns>
02376         public static XmlDocument SaveAsSVG(this Page page, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default, bool useStyles = false)
02377         {
02378             return CreateSVGDocument(page, "", textOption, linkDestinations, filterOption, useStyles,
      true);
02379         }
02380
02381         private static XmlDocument CreateSVGDocument(this Page page, string tagPrefix, TextOptions
      textOption, Dictionary<string, string> linkDestinations, FilterOption filterOption, bool useStyles,
      bool reuseGradients)
02382         {
02383             if (linkDestinations == null)
```

```
02384             {
02385                 linkDestinations = new Dictionary<string, string>();
02386             }
02387 
02388             if (filterOption == null)
02389             {
02390                 filterOption = FilterOption.Default;
02391             }
02392 
02393             bool textToPaths = textOption == TextOptions.ConvertIntoPaths;
02394 
02395             SVGContext ctx = new SVGContext(page.Width, page.Height, textToPaths, textOption,
      linkDestinations, filterOption, useStyles, tagPrefix, reuseGradients);
02396 
02397             ctx.Rectangle(0, 0, page.Width, page.Height);
02398             ctx.SetFillStyle(page.Background);
02399             ctx.Fill(FillRule.NonZeroWinding);
02400             ctx.SetFillStyle((0, 0, 0, 1));
02401 
02402             page.Graphics.CopyToIGraphicsContext(ctx);
02403 
02404             if (!textToPaths && textOption != TextOptions.DoNotEmbed && textOption !=
      TextOptions.ConvertIntoPathsUsingGlyphs)
02405             {
02406                 bool subsetFonts = textOption == TextOptions.SubsetFonts;
02407 
02408                 StringBuilder cssFonts = new StringBuilder();
02409 
02410                 Dictionary<string, string> newFontFamilies = new Dictionary<string, string>();
02411 
02412                 foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02413                 {
02414                     TrueTypeFile subsettedFont;
02415 
02416                     if (subsetFonts)
02417                     {
02418                         newFontFamilies[kvp.Key] = kvp.Value.TrueTypeFile.GetFontFamilyName() + "-" +
      Guid.NewGuid().ToString();
02419                         subsettedFont = kvp.Value.TrueTypeFile.SubsetFont(new
      string(ctx.UsedChars[kvp.Key].ToArray()));
02420                     }
02421                     else
02422                     {
02423                         newFontFamilies[kvp.Key] = kvp.Value.TrueTypeFile.GetFontName();
02424                         subsettedFont = kvp.Value.TrueTypeFile;
02425                     }
02426 
02427                     byte[] fontBytes;
02428 
02429                     using (MemoryStream fontStream = new
      MemoryStream((int)subsettedFont.FontStream.Length))
02430                     {
02431                         subsettedFont.FontStream.Seek(0, SeekOrigin.Begin);
02432                         subsettedFont.FontStream.CopyTo(fontStream);
02433 
02434                         fontBytes = fontStream.ToArray();
02435                     }
02436 
02437 
02438                     cssFonts.Append("\n    @font-face\n    {\n      font-family:  \"" +
      newFontFamilies[kvp.Key] + "\";\n      src:  url(\"data:font/ttf;charset=utf-8;base64,");
02439                     cssFonts.Append(Convert.ToBase64String(fontBytes));
02440                     cssFonts.Append("\");\n    }\n  ");
02441                 }
02442 
02443                 XmlElement style = ctx.Document.CreateElement("style", SVGContext.SVGNamespace);
02444                 style.InnerText = cssFonts.ToString();
02445 
02446                 XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02447 
02448                 svgElement.InsertBefore(style, svgElement.FirstChild);
02449 
02450                 HashSet<string> updatedClasses = new HashSet<string>();
02451 
02452                 foreach (XmlNode text in ctx.Document.GetElementsByTagName("text"))
02453                 {
02454                     string fontFamily;
02455 
02456                     if (!useStyles)
02457                     {
02458                         fontFamily = text.Attributes["font-family"].Value;
02459                     }
02460                     else
02461                     {
02462                         fontFamily = ctx.Styles[text.Attributes["class"].Value]["font-family"];
02463                     }
02464 
```

```
02465                     if (!useStyles || updatedClasses.Add(text.Attributes["class"].Value))
02466                     {
02467
02468                         string fallbackFontFamily = "";
02469
02470                         switch (fontFamily)
02471                         {
02472                             case "Helvetica":
02473                             case "Helvetica-Bold":
02474                             case "Helvetica-Oblique":
02475                             case "Helvetica-BoldOblique":
02476                                 fallbackFontFamily = "sans-serif";
02477                                 break;
02478
02479                             case "Times-Roman":
02480                             case "Times-Bold":
02481                             case "Times-Italic":
02482                             case "Times-BoldItalic":
02483                                 fallbackFontFamily = "serif";
02484                                 break;
02485
02486                             case "Courier":
02487                             case "Courier-Bold":
02488                             case "Courier-Oblique":
02489                             case "Courier-BoldOblique":
02490                                 fallbackFontFamily = "monospace";
02491                                 break;
02492
02493                             default:
02494                                 if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsFixedPitch())
02495                                 {
02496                                     fallbackFontFamily = "monospace";
02497                                 }
02498                                 else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsScript() ||
    ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsSerif())
02499                                 {
02500                                     fallbackFontFamily = "serif";
02501                                 }
02502                                 else
02503                                 {
02504                                     fallbackFontFamily = "sans-serif";
02505                                 }
02506                                 break;
02507                         }
02508
02509                         if (!useStyles)
02510                         {
02511                             if (!string.IsNullOrEmpty(fallbackFontFamily))
02512                             {
02513                                 ((XmlElement)text).SetAttribute("font-family",
    newFontFamilies[fontFamily] + ", " + fallbackFontFamily);
02514                             }
02515                             else
02516                             {
02517                                 ((XmlElement)text).SetAttribute("font-family",
    newFontFamilies[fontFamily]);
02518                             }
02519                         }
02520                         else
02521                         {
02522                             if (!string.IsNullOrEmpty(fallbackFontFamily))
02523                             {
02524                                 ctx.Styles[text.Attributes["class"].Value]["font-family"] =
    newFontFamilies[fontFamily] + ", " + fallbackFontFamily;
02525                             }
02526                             else
02527                             {
02528                                 ctx.Styles[text.Attributes["class"].Value]["font-family"] =
    newFontFamilies[fontFamily];
02529                             }
02530                         }
02531                     }
02532                 }
02533             }
02534             else if (textOption == TextOptions.ConvertIntoPathsUsingGlyphs)
02535             {
02536                 foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02537                 {
02538                     string guid = Guid.NewGuid().ToString();
02539
02540                     XmlElement defs = ctx.Document.CreateElement("defs", SVGContext.SVGNamespace);
02541                     defs.SetAttribute("id", kvp.Value.FamilyName + "-glyphs-" + guid);
02542
02543                     foreach (char c in ctx.UsedChars[kvp.Key])
02544                     {
02545                         XmlElement charPath = ctx.Document.CreateElement("path",
    SVGContext.SVGNamespace);
```

```
02546                         charPath.SetAttribute("id", kvp.Key + "-" + c);
02547
02548                         TrueTypeFile.TrueTypePoint[][] glyphPaths =
      kvp.Value.TrueTypeFile.GetGlyphPath(c, kvp.Value.TrueTypeFile.GetUnitsPerEm());
02549
02550                         StringBuilder data = new StringBuilder();
02551
02552                         for (int j = 0; j < glyphPaths.Length; j++)
02553                         {
02554                             for (int k = 0; k < glyphPaths[j].Length; k++)
02555                             {
02556                                 if (k == 0)
02557                                 {
02558                                     data.Append("M " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02559                                 }
02560                                 else
02561                                 {
02562                                     if (glyphPaths[j][k].IsOnCurve)
02563                                     {
02564                                         data.Append("L " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02565                                     }
02566                                     else
02567                                     {
02568                                         data.Append("Q " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02569                                         data.Append(glyphPaths[j][k +
      1].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (-glyphPaths[j][k +
      1].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02570
02571                                         k++;
02572                                     }
02573                                 }
02574                             }
02575
02576                             data.Append("Z");
02577                         }
02578
02579                         charPath.SetAttribute("d", data.ToString());
02580
02581                         defs.AppendChild(charPath);
02582                     }
02583
02584                     XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02585                     svgElement.InsertBefore(defs, svgElement.FirstChild);
02586                 }
02587             }
02588             else if (!textToPaths && textOption == TextOptions.DoNotEmbed)
02589             {
02590                 HashSet<string> updatedClasses = new HashSet<string>();
02591
02592                 foreach (XmlNode text in ctx.Document.GetElementsByTagName("text"))
02593                 {
02594                     string fontFamily;
02595
02596                     if (!useStyles)
02597                     {
02598                         fontFamily = text.Attributes["font-family"].Value;
02599                     }
02600                     else
02601                     {
02602                         fontFamily = ctx.Styles[text.Attributes["class"].Value]["font-family"];
02603                     }
02604
02605                     if (!useStyles || updatedClasses.Add(text.Attributes["class"].Value))
02606                     {
02607
02608                         string newFontFamily =
      ctx.UsedFontFamilies[fontFamily].TrueTypeFile.GetFontFamilyName();
02609
02610                         switch (fontFamily)
02611                         {
02612                             case "Helvetica":
02613                             case "Helvetica-Bold":
02614                             case "Helvetica-Oblique":
02615                             case "Helvetica-BoldOblique":
02616                                 newFontFamily = newFontFamily + ", sans-serif";
02617                                 break;
02618
02619                             case "Times-Roman":
02620                             case "Times-Bold":
02621                             case "Times-Italic":
02622                             case "Times-BoldItalic":
```

```
02623                                   newFontFamily = newFontFamily + ", serif";
02624                                   break;
02625
02626                               case "Courier":
02627                               case "Courier-Bold":
02628                               case "Courier-Oblique":
02629                               case "Courier-BoldOblique":
02630                                   newFontFamily = newFontFamily + ", monospace";
02631                                   break;
02632
02633                               default:
02634                                   if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsFixedPitch())
02635                                   {
02636                                       newFontFamily = newFontFamily + ", monospace";
02637                                   }
02638                                   else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsScript())
02639                                   {
02640                                       newFontFamily = newFontFamily + ", cursive";
02641                                   }
02642                                   else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsSerif())
02643                                   {
02644                                       newFontFamily = newFontFamily + ", serif";
02645                                   }
02646                                   else
02647                                   {
02648                                       newFontFamily = newFontFamily + ", sans-serif";
02649                                   }
02650                                   break;
02651                           }
02652
02653                           if (!useStyles)
02654                           {
02655                               ((XmlElement)text).SetAttribute("font-family", newFontFamily);
02656                           }
02657                           else
02658                           {
02659                               ctx.Styles[text.Attributes["class"].Value]["font-family"] = newFontFamily;
02660                           }
02661                       }
02662                   }
02663               }
02664
02665               if (useStyles)
02666               {
02667                   StringBuilder classStyle = new StringBuilder();
02668
02669                   if (ctx.Styles.Count > 0)
02670                   {
02671                       classStyle.Append("\n  ");
02672                   }
02673
02674                   foreach (KeyValuePair<string, Dictionary<string, string» style in ctx.Styles)
02675                   {
02676                       classStyle.Append("  .");
02677                       classStyle.Append(style.Key);
02678                       classStyle.Append("\n");
02679                       classStyle.Append("    {\n");
02680
02681                       foreach (KeyValuePair<string, string> kvp in style.Value)
02682                       {
02683                           classStyle.Append("      ");
02684                           classStyle.Append(kvp.Key);
02685                           classStyle.Append(":  ");
02686                           classStyle.Append(kvp.Value);
02687                           classStyle.Append(";\n");
02688                       }
02689
02690                       classStyle.Append("    }\n  ");
02691                   }
02692
02693
02694                   XmlElement styleElement = ctx.Document.CreateElement("style",
      SVGContext.SVGNamespace);
02695                   styleElement.InnerText = classStyle.ToString();
02696
02697                   XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02698
02699                   svgElement.InsertBefore(styleElement, svgElement.FirstChild);
02700               }
02701
02702
02703
02704               ctx.Document.DocumentElement.SetAttribute("style", "font-synthesis:  none;");
02705
02706               return ctx.Document;
02707           }
02708
```

```
02709 /// <summary>
02710 /// Render the animation to an SVG document, using SVG animations.
02711 /// </summary>
02712 /// <param name="animation">The <see cref="Animation"/> to render.</param>
02713 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
02714 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
02715 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02716 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02717 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02718 /// <returns>An <see cref="XmlDocument"/> containing the animated SVG image.</returns>
02719         public static XmlDocument SaveAsAnimatedSVG(this Animation animation, bool includeControls =
      false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts,
      Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
02720         {
02721             (XmlDocument, double)[] frames = new (XmlDocument, double)[animation.Frames.Count];
02722             double[] durations = new double[animation.Frames.Count];
02723 
02724             (XmlDocument, XmlDocument, Transition)[] transitions = new (XmlDocument, XmlDocument,
      Transition)[animation.Frames.Count - 1];
02725 
02726 
02727             string repeatCount = animation.RepeatCount <= 0 ?  "indefinite" :
      animation.RepeatCount.ToString();
02728 
02729             double currentTime = 0;
02730 
02731             for (int i = 0; i < frames.Length; i++)
02732             {
02733                 Page pag = new Page(animation.Width, animation.Height) { Background =
      animation.Background };
02734                 pag.Graphics.DrawGraphics(0, 0, animation.Frames[i].Graphics);
02735 
02736                 if (i > 0)
02737                 {
02738                     frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption,
      linkDestinations, filterOption, false, false), animation.Frames[i].Duration);
02739                     durations[i] = animation.Frames[i].Duration * durationScaling;
02740 
02741                     if (animation.Transitions[i - 1].Duration > 0)
02742                     {
02743                         double epsilon = 1e-5 * animation.Transitions[i - 1].Duration;
02744 
02745                         Page startPage = animation.GetFrameAtAbsolute(currentTime + epsilon);
02746                         Page endPage = animation.GetFrameAtAbsolute(currentTime +
      animation.Transitions[i - 1].Duration - epsilon);
02747 
02748                         transitions[i - 1] = (startPage.CreateSVGDocument("transition" + i.ToString()
      + "://", textOption, linkDestinations, filterOption, false, false),
      endPage.CreateSVGDocument("transition" + i.ToString() + "://", textOption, linkDestinations,
      filterOption, false, false), animation.Transitions[i - 1]);
02749                     }
02750                     else
02751                     {
02752                         transitions[i - 1] = (null, null, animation.Transitions[i - 1]);
02753                     }
02754 
02755                     currentTime += animation.Frames[i].Duration + animation.Transitions[i -
      1].Duration;
02756                 }
02757                 else
02758                 {
02759                     frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption,
      linkDestinations, filterOption, false, false), animation.Frames[i].Duration);
02760                     durations[i] = animation.Frames[i].Duration * durationScaling;
02761                     currentTime += animation.Frames[i].Duration;
02762                 }
02763             }
02764 
02765             double totalDuration = animation.Duration * durationScaling;
02766 
02767             XmlDocument Document = new XmlDocument();
02768 
02769             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null),
      Document.DocumentElement);
02770 
02771             XmlElement currentElement = Document.CreateElement(null, "svg", SVGContext.SVGNamespace);
02772             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
02773             currentElement.SetAttribute("viewBox", "0 0 " +
```

```
    animation.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    animation.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02774             currentElement.SetAttribute("version", "1.1");
02775             Document.AppendChild(currentElement);
02776
02777             currentTime = 0;
02778
02779             for (int i = 0; i < frames.Length; i++)
02780             {
02781                 if (i > 0 && transitions[i - 1].Item3.Duration > 0)
02782                 {
02783                     XmlNode clonedSVG = Document.ImportNode(transitions[i -
    1].Item1.GetElementsByTagName("svg")[0], true);
02784                     XmlNode clonedG = Document.CreateElement("g", SVGContext.SVGNamespace);
02785                     clonedG.InnerXml = clonedSVG.InnerXml;
02786                     clonedSVG = clonedG;
02787
02788                     {
02789                         XmlElement animate = Document.CreateElement("animate",
    SVGContext.SVGNamespace);
02790                         animate.SetAttribute("attributeName", "display");
02791                         animate.SetAttribute("values", "none;none;block;block;none;none");
02792                         animate.SetAttribute("dur",
    totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02793                         animate.SetAttribute("repeatCount", repeatCount);
02794                         animate.SetAttribute("fill", "freeze");
02795                         animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
    Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02796                         clonedSVG.InsertBefore(animate, clonedSVG.FirstChild);
02797                     }
02798
02799                     currentElement.AppendChild(clonedSVG);
02800
02801                     Dictionary<string, XmlElement> taggedStart = GetTaggedElements(clonedSVG);
02802                     Dictionary<string, XmlElement> taggedEnd = GetTaggedElements(transitions[i -
    1].Item2.GetElementsByTagName("svg")[0]);
02803
02804                     string overallEasing = null;
02805
02806                     if (transitions[i - 1].Item3.OverallEasing is SplineEasing spline)
02807                     {
02808                         overallEasing = "0.5 0.5 0.5 0.5; " +
    spline.ControlPoint1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    spline.ControlPoint1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
02809
    spline.ControlPoint2.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    spline.ControlPoint2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + "; 0.5 0.5 0.5
    0.5";
02810                     }
02811
02812                     foreach (KeyValuePair<string, XmlElement> kvp in taggedStart)
02813                     {
02814                         if (taggedEnd.TryGetValue(kvp.Key, out XmlElement endElement))
02815                         {
02816                             List<(string, string, string)> attributesToChange = new List<(string,
    string, string)>();
02817
02818                             foreach (XmlAttribute attr in kvp.Value.Attributes)
02819                             {
02820                                 if (endElement.HasAttribute(attr.Name))
02821                                 {
02822                                     string oldValue = attr.Value;
02823                                     string newValue = endElement.GetAttribute(attr.Name);
02824
02825                                     if (oldValue != newValue)
02826                                     {
02827                                         attributesToChange.Add((attr.Name, oldValue, newValue));
02828                                     }
02829                                 }
02830                             }
02831
02832                             string easing = overallEasing;
02833
02834                             string itemId = kvp.Key.Substring(kvp.Key.IndexOf("://") + 3);
02835
02836                             if (kvp.Value.Name == "linearGradient" || kvp.Value.Name ==
    "radialGradient")
02837                             {
02838                                 itemId = itemId.Substring(itemId.IndexOf("_") + 1);
02839                             }
02840                             else if (kvp.Value.Name == "stop")
02841                             {
```

```
02842                                 itemId = itemId.Substring(itemId.IndexOf("_") + 1);
02843                                 itemId = itemId.Substring(0, itemId.IndexOf("/"));
02844                             }
02845 
02846                             if (transitions[i - 1].Item3.Easings != null && transitions[i -
      1].Item3.Easings.TryGetValue(itemId, out IEasing currEasing) && currEasing is SplineEasing currSpline)
02847                             {
02848                                 easing = "0.5 0.5 0.5 0.5; " +
      currSpline.ControlPoint1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      currSpline.ControlPoint1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
02849 
      currSpline.ControlPoint2.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      currSpline.ControlPoint2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + "; 0.5 0.5
      0.5 0.5";
02850                             }
02851 
02852                             for (int j = 0; j < attributesToChange.Count; j++)
02853                             {
02854                                 kvp.Value.RemoveAttribute(attributesToChange[j].Item1);
02855 
02856                                 if (attributesToChange[j].Item1 != "transform")
02857                                 {
02858                                     XmlElement animate = Document.CreateElement("animate",
      SVGContext.SVGNamespace);
02859                                     animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
02860                                     animate.SetAttribute("values", attributesToChange[j].Item2 + ";" +
      attributesToChange[j].Item2 + ";" + attributesToChange[j].Item3 + ";" + attributesToChange[j].Item3);
02861                                     animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02862                                     animate.SetAttribute("repeatCount", repeatCount);
02863                                     animate.SetAttribute("fill", "freeze");
02864                                     animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02865 
02866                                     if (!string.IsNullOrEmpty(easing))
02867                                     {
02868                                         animate.SetAttribute("calcMode", "spline");
02869                                         animate.SetAttribute("keySplines", easing);
02870                                     }
02871 
02872                                     if (kvp.Value.HasChildNodes)
02873                                     {
02874                                         kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
02875                                     }
02876                                     else
02877                                     {
02878                                         kvp.Value.AppendChild(animate);
02879                                     }
02880                                 }
02881                                 else
02882                                 {
02883                                     if (attributesToChange[j].Item2.StartsWith("matrix") &&
      attributesToChange[j].Item3.StartsWith("matrix"))
02884                                     {
02885                                         (Point, double, Size, Size) transformStart =
      DecomposeMatrix(attributesToChange[j].Item2);
02886                                         (Point, double, Size, Size) transformEnd =
      DecomposeMatrix(attributesToChange[j].Item3);
02887 
02888                                         if (transformStart.Item4.Height != 0 ||
      transformEnd.Item4.Height != 0)
02889                                         {
02890                                             string skewStart =
      transformStart.Item4.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
02891                                             string skewEnd =
      transformEnd.Item4.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
02892 
02893                                             XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
02894                                             animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
02895                                             animate.SetAttribute("type", "skewY");
02896                                             animate.SetAttribute("values", skewStart + ";" + skewStart
      + ";" + skewEnd + ";" + skewEnd);
02897                                             animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02898                                             animate.SetAttribute("repeatCount", repeatCount);
02899                                             animate.SetAttribute("fill", "freeze");
02900                                             animate.SetAttribute("additive", "sum");
02901                                             animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02902                                             if (!string.IsNullOrEmpty(easing))
```

```
02903                                           {
02904                                               animate.SetAttribute("calcMode", "spline");
02905                                               animate.SetAttribute("keySplines", easing);
02906                                           }
02907
02908                                           if (kvp.Value.HasChildNodes)
02909                                           {
02910                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
02911                                           }
02912                                           else
02913                                           {
02914                                               kvp.Value.AppendChild(animate);
02915                                           }
02916                                       }
02917
02918                                       if (transformStart.Item4.Width != 0 ||
      transformEnd.Item4.Width != 0)
02919                                       {
02920                                           string skewStart =
      transformStart.Item4.Width.ToString(System.Globalization.CultureInfo.InvariantCulture);
02921                                           string skewEnd =
      transformEnd.Item4.Width.ToString(System.Globalization.CultureInfo.InvariantCulture);
02922
02923                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
02924                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
02925                                           animate.SetAttribute("type", "skewX");
02926                                           animate.SetAttribute("values", skewStart + ";" + skewStart
      + ";" + skewEnd + ";" + skewEnd);
02927                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02928                                           animate.SetAttribute("repeatCount", repeatCount);
02929                                           animate.SetAttribute("fill", "freeze");
02930                                           animate.SetAttribute("additive", "sum");
02931                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02932                                           if (!string.IsNullOrEmpty(easing))
02933                                           {
02934                                               animate.SetAttribute("calcMode", "spline");
02935                                               animate.SetAttribute("keySplines", easing);
02936                                           }
02937
02938                                           if (kvp.Value.HasChildNodes)
02939                                           {
02940                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
02941                                           }
02942                                           else
02943                                           {
02944                                               kvp.Value.AppendChild(animate);
02945                                           }
02946                                       }
02947
02948                                       if (transformStart.Item3.Width != 0 ||
      transformStart.Item3.Height != 0 || transformEnd.Item3.Width != 0 || transformEnd.Item3.Height != 0)
02949                                       {
02950                                           string scaleStart =
      transformStart.Item3.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformStart.Item3.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
02951                                           string scaleEnd =
      transformEnd.Item3.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformEnd.Item3.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
02952
02953                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
02954                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
02955                                           animate.SetAttribute("type", "scale");
02956                                           animate.SetAttribute("values", scaleStart + ";" +
      scaleStart + ";" + scaleEnd + ";" + scaleEnd);
02957                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02958                                           animate.SetAttribute("repeatCount", repeatCount);
02959                                           animate.SetAttribute("fill", "freeze");
02960                                           animate.SetAttribute("additive", "sum");
02961                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02962                                           if (!string.IsNullOrEmpty(easing))
02963                                           {
02964                                               animate.SetAttribute("calcMode", "spline");
02965                                               animate.SetAttribute("keySplines", easing);
02966                                           }
02967
```

```
02968                                           if (kvp.Value.HasChildNodes)
02969                                           {
02970                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
02971                                           }
02972                                           else
02973                                           {
02974                                               kvp.Value.AppendChild(animate);
02975                                           }
02976                                       }
02977
02978                                       if (transformStart.Item2 != 0 || transformEnd.Item2 != 0)
02979                                       {
02980                                           string rotateStart = (transformStart.Item2 * 180 /
      Math.PI).ToString(System.Globalization.CultureInfo.InvariantCulture);
02981                                           string rotateEnd = (transformEnd.Item2 * 180 /
      Math.PI).ToString(System.Globalization.CultureInfo.InvariantCulture);
02982
02983                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
02984                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
02985                                           animate.SetAttribute("type", "rotate");
02986                                           animate.SetAttribute("values", rotateStart + ";" +
      rotateStart + ";" + rotateEnd + ";" + rotateEnd);
02987                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
02988                                           animate.SetAttribute("repeatCount", repeatCount);
02989                                           animate.SetAttribute("fill", "freeze");
02990                                           animate.SetAttribute("additive", "sum");
02991                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
02992                                           if (!string.IsNullOrEmpty(easing))
02993                                           {
02994                                               animate.SetAttribute("calcMode", "spline");
02995                                               animate.SetAttribute("keySplines", easing);
02996                                           }
02997
02998                                           if (kvp.Value.HasChildNodes)
02999                                           {
03000                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03001                                           }
03002                                           else
03003                                           {
03004                                               kvp.Value.AppendChild(animate);
03005                                           }
03006                                       }
03007
03008                                       if (transformStart.Item1.X != 0 || transformStart.Item1.Y != 0
      || transformEnd.Item1.X != 0 || transformEnd.Item1.Y != 0)
03009                                       {
03010                                           string translateStart =
      transformStart.Item1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformStart.Item1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
03011                                           string translateEnd =
      transformEnd.Item1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformEnd.Item1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
03012
03013                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03014                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03015                                           animate.SetAttribute("type", "translate");
03016                                           animate.SetAttribute("values", translateStart + ";" +
      translateStart + ";" + translateEnd + ";" + translateEnd);
03017                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03018                                           animate.SetAttribute("repeatCount", repeatCount);
03019                                           animate.SetAttribute("fill", "freeze");
03020                                           animate.SetAttribute("additive", "sum");
03021                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03022                                           if (!string.IsNullOrEmpty(easing))
03023                                           {
03024                                               animate.SetAttribute("calcMode", "spline");
03025                                               animate.SetAttribute("keySplines", easing);
03026                                           }
03027
03028                                           if (kvp.Value.HasChildNodes)
03029                                           {
03030                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03031                                           }
03032                                           else
03033                                           {
```

```
03034                                                   kvp.Value.AppendChild(animate);
03035                                               }
03036                                           }
03037                                       }
03038                                       else
03039                                       {
03040                                           throw new NotImplementedException("Invalid transform!");
03041                                       }
03042                                   }
03043                               }
03044                           }
03045                       }
03046
03047                       currentTime += transitions[i - 1].Item3.Duration * durationScaling;
03048                   }
03049
03050                   {
03051                       XmlNode clonedSVG =
      Document.ImportNode(frames[i].Item1.GetElementsByTagName("svg")[0], true);
03052
03053                       XmlElement animate = Document.CreateElement("animate", SVGContext.SVGNamespace);
03054                       animate.SetAttribute("attributeName", "display");
03055                       animate.SetAttribute("values", "none;none;block;block;none;none");
03056                       animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03057                       animate.SetAttribute("repeatCount", repeatCount);
03058
03059                       if (i < frames.Length - 1)
03060                       {
03061                           animate.SetAttribute("fill", "freeze");
03062                       }
03063
03064                       animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + durations[i]) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      (currentTime + durations[i]) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03065                       clonedSVG.InsertBefore(animate, clonedSVG.FirstChild);
03066
03067                       currentElement.AppendChild(clonedSVG);
03068                   }
03069
03070                   currentTime += durations[i];
03071               }
03072
03073               if (includeControls)
03074               {
03075                   double controlsHeight = animation.Height * 0.08;
03076
03077                   double controlsWidth = animation.Width * 0.8;
03078
03079                   if (controlsWidth < controlsHeight * 5)
03080                   {
03081                       controlsHeight = controlsWidth / 5;
03082                   }
03083
03084                   currentElement.AppendChild(Document.ImportNode(GetAnimationControls(animation.Width,
      animation.Height, controlsWidth, controlsHeight, animation.Duration * durationScaling,
      animation.RepeatCount).GetElementsByTagName("svg")[0], true));
03085               }
03086
03087               return Document;
03088           }
03089
03090           private static XmlDocument GetAnimationControls(double width, double height, double
      controlsWidth, double controlsHeight, double totalDuration, int repeatCount)
03091           {
03092               double controlsMargin = controlsHeight / 12;
03093               double controlRadius = controlsHeight / 6;
03094
03095               Page pag = new Page(width, height);
03096
03097               Graphics gpr = pag.Graphics;
03098
03099               GraphicsPath background = new GraphicsPath().MoveTo(width * 0.5 - controlsWidth * 0.5 +
      controlRadius, height - controlsMargin - controlsHeight).LineTo(width * 0.5 + controlsWidth * 0.5 -
      controlRadius, height - controlsMargin - controlsHeight);
03100               background.Arc(width * 0.5 + controlsWidth * 0.5 - controlRadius, height - controlsMargin
      - controlsHeight + controlRadius, controlRadius, -Math.PI / 2, 0).LineTo(width * 0.5 + controlsWidth *
      0.5, height - controlsMargin - controlRadius);
03101               background.Arc(width * 0.5 + controlsWidth * 0.5 - controlRadius, height - controlsMargin
      - controlRadius, controlRadius, 0, Math.PI / 2).LineTo(width * 0.5 - controlsWidth * 0.5 +
      controlRadius, height - controlsMargin);
03102               background.Arc(width * 0.5 - controlsWidth * 0.5 + controlRadius, height - controlsMargin
      - controlRadius, controlRadius, Math.PI / 2, Math.PI).LineTo(width * 0.5 - controlsWidth * 0.5, height
```

```
    - controlsMargin - controlsHeight + controlRadius);
03103             background.Arc(width * 0.5 - controlsWidth * 0.5 + controlRadius, height - controlsMargin
    - controlsHeight + controlRadius, controlRadius, Math.PI, 3 * Math.PI / 2).Close();
03104
03105             gpr.FillPath(background, Colour.FromRgba(0, 0, 0, 0.5), tag:  "timeLineBackground");
03106
03107             double roundedCornerSize = 0.25;
03108
03109             LinearGradientBrush buttonBrush = new LinearGradientBrush(new Point(0, controlsHeight),
    new Point(0, 0), new GradientStop(Colours.White, 1), new GradientStop(Colour.FromRgb(220, 220, 220),
    0));
03110
03111             gpr.Save();
03112             gpr.Translate(width * 0.5 - controlsWidth * 0.5 + controlsHeight * 0.5, height -
    controlsHeight - controlsMargin);
03113             GraphicsPath playButton = new GraphicsPath().MoveTo(0, controlsHeight * 0.2 +
    controlsHeight * 0.6 * roundedCornerSize).LineTo(0, controlsHeight * 0.2 + controlsHeight * 0.6 * (1 -
    roundedCornerSize)).CubicBezierTo(0, controlsHeight * 0.8, 0, controlsHeight * 0.8, controlsHeight *
    0.5 * roundedCornerSize, controlsHeight * 0.5 + controlsHeight * 0.3 * (1 -
    roundedCornerSize)).LineTo(controlsHeight * 0.5 * (1 - roundedCornerSize), controlsHeight * 0.5 +
    controlsHeight * 0.3 * roundedCornerSize);
03114             playButton.CubicBezierTo(controlsHeight * 0.5, controlsHeight * 0.5, controlsHeight * 0.5,
    controlsHeight * 0.5, controlsHeight * 0.5 * (1 - roundedCornerSize), controlsHeight * 0.5 -
    controlsHeight * 0.3 * roundedCornerSize).LineTo(controlsHeight * 0.5 * roundedCornerSize,
    controlsHeight * 0.5 - controlsHeight * 0.3 * (1 - roundedCornerSize));
03115             playButton.CubicBezierTo(0, controlsHeight * 0.2, 0, controlsHeight * 0.2, 0,
    controlsHeight * 0.2 + controlsHeight * 0.6 * roundedCornerSize).Close();
03116             gpr.FillPath(playButton, buttonBrush, "playButton");
03117
03118             GraphicsPath pauseButton = new GraphicsPath().MoveTo(controlsHeight * 0.2 *
    roundedCornerSize, controlsHeight * 0.25).LineTo(controlsHeight * 0.2 * (1 - roundedCornerSize),
    controlsHeight * 0.25).CubicBezierTo(controlsHeight * 0.2, controlsHeight * 0.25, controlsHeight *
    0.2, controlsHeight * 0.25, controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 *
    roundedCornerSize);
03119             pauseButton.LineTo(controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 * (1
    - roundedCornerSize)).CubicBezierTo(controlsHeight * 0.2, controlsHeight * 0.75, controlsHeight * 0.2,
    controlsHeight * 0.75, controlsHeight * 0.2 * (1 - roundedCornerSize), controlsHeight *
    0.75).LineTo(controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.75);
03120             pauseButton.CubicBezierTo(0, controlsHeight * 0.75, 0, controlsHeight * 0.75, 0,
    controlsHeight * 0.25 + controlsHeight * 0.5 * (1 - roundedCornerSize)).LineTo(0, controlsHeight *
    0.25 + controlsHeight * 0.5 * roundedCornerSize).CubicBezierTo(0, controlsHeight * 0.25, 0,
    controlsHeight * 0.25, controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.25).Close();
03121             pauseButton.MoveTo(controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize,
    controlsHeight * 0.25).LineTo(controlsHeight * 0.3 + controlsHeight * 0.2 * (1 - roundedCornerSize),
    controlsHeight * 0.25).CubicBezierTo(controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight *
    0.25, controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.25, controlsHeight * 0.3 +
    controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 * roundedCornerSize);
03122             pauseButton.LineTo(controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.25 +
    controlsHeight * 0.5 * (1 - roundedCornerSize)).CubicBezierTo(controlsHeight * 0.3 + controlsHeight *
    0.2, controlsHeight * 0.75, controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.75,
    controlsHeight * 0.3 + controlsHeight * 0.2 * (1 - roundedCornerSize), controlsHeight *
    0.75).LineTo(controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.75);
03123             pauseButton.CubicBezierTo(controlsHeight * 0.3, controlsHeight * 0.75, controlsHeight *
    0.3, controlsHeight * 0.75, controlsHeight * 0.3, controlsHeight * 0.25 + controlsHeight * 0.5 * (1 -
    roundedCornerSize)).LineTo(controlsHeight * 0.3, controlsHeight * 0.25 + controlsHeight * 0.5 *
    roundedCornerSize).CubicBezierTo(controlsHeight * 0.3, controlsHeight * 0.25, controlsHeight * 0.3,
    controlsHeight * 0.25, controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize, controlsHeight
    * 0.25).Close();
03124             gpr.FillPath(pauseButton, buttonBrush, "pauseButton");
03125
03126             gpr.FillRectangle(0, controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.6,
    Colour.FromRgba(0, 0, 0, 0), tag:  "playButtonHitbox");
03127             gpr.FillRectangle(0, controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.6,
    Colour.FromRgba(0, 0, 0, 0), tag:  "pauseButtonHitbox");
03128
03129             double timeLineStartX = controlsHeight * 0.5 + controlsHeight * 0.5 + controlsHeight / 3;
03130
03131             gpr.Translate(timeLineStartX, 0);
03132
03133             GraphicsPath timeLine = new GraphicsPath().MoveTo(controlsHeight * 0.1, controlsHeight *
    0.4).LineTo(controlsWidth - timeLineStartX - controlsHeight - controlsHeight * 0.1 - controlsHeight /
    3, controlsHeight * 0.4);
03134             timeLine.Arc(controlsWidth - timeLineStartX - controlsHeight - controlsHeight * 0.1 -
    controlsHeight / 3, controlsHeight * 0.5, controlsHeight * 0.1, -Math.PI / 2, Math.PI /
    2).LineTo(controlsHeight * 0.1, controlsHeight * 0.6);
03135             timeLine.Arc(controlsHeight * 0.1, controlsHeight * 0.5, controlsHeight * 0.1, Math.PI /
    2, 3 * Math.PI / 2).Close();
03136
03137             gpr.FillPath(timeLine, new LinearGradientBrush(new Point(0, controlsHeight * 0.6), new
    Point(0, controlsHeight * 0.4), new GradientStop(Colour.FromRgba(255, 255, 255, 80), 0), new
    GradientStop(Colour.FromRgba(255, 255, 255, 40), 1)), tag:  "timeLine");
03138             gpr.StrokePath(timeLine, Colour.FromRgba(255, 255, 255, 180), tag:  "timeLineStroke");
03139
03140             gpr.Restore();
03141
03142             GraphicsPath thumbPath = new GraphicsPath().MoveTo(-controlsHeight * 0.2, controlsHeight *
    0.3).LineTo(controlsHeight * 0.2, controlsHeight * 0.3);
```

```
03143             thumbPath.Arc(controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.2, -Math.PI /
      2, Math.PI / 2).LineTo(-controlsHeight * 0.2, controlsHeight * 0.7);
03144             thumbPath.Arc(-controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.2, Math.PI /
      2, 3 * Math.PI / 2).Close();
03145
03146             Graphics thumbStart = new Graphics();
03147             thumbStart.Translate(width * 0.5 - controlsWidth * 0.5 + controlsHeight * 0.5 +
      timeLineStartX, height - controlsHeight - controlsMargin);
03148             thumbStart.FillPath(thumbPath, buttonBrush, tag:  "thumb");
03149
03150             Graphics thumbEnd = new Graphics();
03151             thumbEnd.Translate(width * 0.5 + controlsWidth * 0.5 - controlsHeight * 0.5 -
      controlsHeight * 0.4, height - controlsHeight - controlsMargin);
03152             thumbEnd.FillPath(thumbPath, buttonBrush, tag:  "thumb");
03153
03154             Animation thumbAnimation = new Animation(width, height, 1) { RepeatCount = repeatCount };
03155             thumbAnimation.AddFrame(new Frame(thumbStart, 0));
03156             thumbAnimation.AddFrame(new Frame(thumbEnd, 0), new Transition(totalDuration));
03157
03158             XmlDocument animationDoc = thumbAnimation.SaveAsAnimatedSVG();
03159
03160             XmlNodeList svgNodes = animationDoc.GetElementsByTagName("svg");
03161
03162             for (int i = 1; i < svgNodes.Count; i++)
03163             {
03164
      ((XmlElement)svgNodes[i]).RemoveChild(((XmlElement)svgNodes[i]).GetElementsByTagName("path")[0]);
03165             }
03166
03167             ((XmlElement)animationDoc.GetElementsByTagName("svg")[0]).SetAttribute("style",
      "pointer-events:  none");
03168
03169             XmlDocument doc = pag.SaveAsSVG();
03170
03171             XmlNodeList paths = doc.GetElementsByTagName("path");
03172
03173             for (int i = 0; i < paths.Count; i++)
03174             {
03175                 if (((XmlElement)paths[i]).GetAttribute("id") == "timeLine")
03176                 {
03177                     ((XmlElement)paths[i]).SetAttribute("style", "cursor:  pointer;");
03178                 }
03179
03180                 if (((XmlElement)paths[i]).GetAttribute("id") == "playButtonHitbox" ||
      ((XmlElement)paths[i]).GetAttribute("id") == "pauseButtonHitbox")
03181                 {
03182                     ((XmlElement)paths[i]).SetAttribute("style", "cursor:  pointer");
03183                 }
03184             }
03185
03186             XmlNode animationNode = doc.ImportNode(animationDoc.GetElementsByTagName("svg")[0], true);
03187             doc.GetElementsByTagName("svg")[0].AppendChild(animationNode);
03188             ((XmlElement)doc.GetElementsByTagName("svg")[0]).SetAttribute("id", "animationControls");
03189
03190             ((XmlElement)doc.GetElementsByTagName("svg")[0]).SetAttribute("style", "font-synthesis:
      none; transition:  opacity 500ms;");
03191
03192             using (Stream sr =
      Assembly.GetExecutingAssembly().GetManifestResourceStream("VectSharp.SVG.AnimationControls.js"))
03193             using (StreamReader reader = new StreamReader(sr))
03194             {
03195                 string javascript = reader.ReadToEnd();
03196                 javascript = javascript.Replace("@@totalLength@@", (totalDuration /
      1000).ToString(System.Globalization.CultureInfo.InvariantCulture));
03197
03198                 XmlElement script = doc.CreateElement("script", SVGContext.SVGNamespace);
03199                 script.SetAttribute("type", "text/javascript");
03200                 script.InnerText = javascript;
03201
03202                 doc.GetElementsByTagName("svg")[0].AppendChild(script);
03203             }
03204
03205             return doc;
03206         }
03207
03208 /// <summary>
03209 /// Render the animation to an SVG stream, using SVG animations.
03210 /// </summary>
03211 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03212 /// <param name="stream">The <see cref="Stream"/> on which the SVG document will be written.</param>
03213 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
03214 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
03215 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
```

```
03216 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
03217 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
03218         public static void SaveAsAnimatedSVG(this Animation animation, Stream stream, bool
      includeControls = false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts,
      Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
03219         {
03220             XmlDocument document = SaveAsAnimatedSVG(animation, includeControls, durationScaling,
      textOption, linkDestinations, filterOption);
03221             WriteXMLToStream(document.DocumentElement, stream);
03222         }
03223
03224 /// <summary>
03225 /// Render the animation to an SVG file, using SVG animations.
03226 /// </summary>
03227 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03228 /// <param name="fileName">The output file that will be created.</param>
03229 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
03230 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
03231 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03232 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
03233 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
03234         public static void SaveAsAnimatedSVG(this Animation animation, string fileName, bool
      includeControls = false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts,
      Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
03235         {
03236             using (FileStream fs = File.Create(fileName))
03237             {
03238                 SaveAsAnimatedSVG(animation, fs, includeControls, durationScaling, textOption,
      linkDestinations, filterOption);
03239             }
03240         }
03241
03242 /// <summary>
03243 /// Render the animation to an SVG document, encoding discrete frames.
03244 /// </summary>
03245 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03246 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
03247 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
      (fps).</param>
03248 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
03249 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03250 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
03251 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
03252 /// <returns>An <see cref="XmlDocument"/> containing the animated SVG image.</returns>
03253         public static XmlDocument SaveAsAnimatedSVGWithFrames(this Animation animation, bool
      includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default)
03254         {
03255             int frameCount = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling /
      1000);
03256
03257             (XmlDocument, double)[] frames = new (XmlDocument, double)[frameCount];
03258
03259             Parallel.For(0, frameCount, i =>
03260             {
03261                 double frameTime = i / frameRate / durationScaling * 1000;
03262
03263                 double frameDuration = Math.Min((animation.Duration - frameTime) * durationScaling,
      1000 / frameRate);
03264
03265                 Page pag = animation.GetFrameAtAbsolute(frameTime);
03266
03267                 frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption,
```

```
    linkDestinations, filterOption, false, true), frameDuration);
03268             });
03269
03270             string repeatCount = animation.RepeatCount <= 0 ?  "indefinite" :
    animation.RepeatCount.ToString();
03271
03272             double totalDuration = animation.Duration * durationScaling;
03273
03274             XmlDocument Document = new XmlDocument();
03275
03276             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null),
    Document.DocumentElement);
03277
03278             XmlElement currentElement = Document.CreateElement(null, "svg", SVGContext.SVGNamespace);
03279             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
03280             currentElement.SetAttribute("viewBox", "0 0 " +
    animation.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    animation.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
03281             currentElement.SetAttribute("version", "1.1");
03282             Document.AppendChild(currentElement);
03283
03284             double currentTime = 0;
03285
03286             for (int i = 0; i < frames.Length; i++)
03287             {
03288                 XmlNode clonedSVG =
    Document.ImportNode(frames[i].Item1.GetElementsByTagName("svg")[0], true);
03289                 XmlNode clonedG = Document.CreateElement("g", SVGContext.SVGNamespace);
03290                 clonedG.InnerXml = clonedSVG.InnerXml;
03291
03292                 XmlElement animate = Document.CreateElement("animate", SVGContext.SVGNamespace);
03293                 animate.SetAttribute("attributeName", "display");
03294                 animate.SetAttribute("values", "none;none;block;block;none;none");
03295                 animate.SetAttribute("dur",
    totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03296                 animate.SetAttribute("repeatCount", repeatCount);
03297                 animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
    Math.Min(1, (currentTime + frames[i].Item2) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    (currentTime + frames[i].Item2) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03298                 clonedG.InsertBefore(animate, clonedG.FirstChild);
03299
03300                 currentElement.AppendChild(clonedG);
03301
03302                 currentTime += frames[i].Item2;
03303             }
03304
03305             if (includeControls)
03306             {
03307                 double controlsHeight = animation.Height * 0.08;
03308
03309                 double controlsWidth = animation.Width * 0.8;
03310
03311                 if (controlsWidth < controlsHeight * 5)
03312                 {
03313                     controlsHeight = controlsWidth / 5;
03314                 }
03315
03316                 currentElement.AppendChild(Document.ImportNode(GetAnimationControls(animation.Width,
    animation.Height, controlsWidth, controlsHeight, animation.Duration * durationScaling,
    animation.RepeatCount).GetElementsByTagName("svg")[0], true));
03317             }
03318
03319             return Document;
03320         }
03321
03322 /// <summary>
03323 /// Render the animation to an SVG stream, encoding discrete frames.
03324 /// </summary>
03325 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03326 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
    contain playback controls that use Javascript to play/pause the animation and change the current
    time.</param>
03327 /// <param name="stream">The <see cref="Stream"/> on which the SVG document will be written.</param>
03328 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
    (fps).</param>
03329 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
    animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
    that this does not affect the frame rate of the animation.</param>
03330 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03331 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
    provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
    the destination specified in the dictionary.  If the destination starts with a hash (#), it is
    interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
```

```
external URI.</param>
03332 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03333         public static void SaveAsAnimatedSVGWithFrames(this Animation animation, Stream stream, bool
includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
= default)
03334         {
03335             XmlDocument document = SaveAsAnimatedSVGWithFrames(animation, includeControls, frameRate,
durationScaling, textOption, linkDestinations, filterOption);
03336             WriteXMLToStream(document.DocumentElement, stream);
03337         }
03338
03339 /// <summary>
03340 /// Render the animation to an SVG file, encoding discrete frames.
03341 /// </summary>
03342 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03343 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
contain playback controls that use Javascript to play/pause the animation and change the current
time.</param>
03344 /// <param name="fileName">The output file that will be created.</param>
03345 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
(fps).</param>
03346 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
that this does not affect the frame rate of the animation.</param>
03347 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03348 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
the destination specified in the dictionary.  If the destination starts with a hash (#), it is
interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
external URI.</param>
03349 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03350         public static void SaveAsAnimatedSVGWithFrames(this Animation animation, string fileName, bool
includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
= default)
03351         {
03352             using (FileStream fs = File.Create(fileName))
03353             {
03354                 SaveAsAnimatedSVGWithFrames(animation, fs, includeControls, frameRate,
durationScaling, textOption, linkDestinations, filterOption);
03355             }
03356         }
03357
03358         // Adapted from https://math.stackexchange.com/a/2888105
03359         private static (Point, double, Size, Size) DecomposeMatrix(string transformMatrix)
03360         {
03361             transformMatrix = transformMatrix.Trim().Substring(7);
03362             transformMatrix = transformMatrix.Substring(0, transformMatrix.Length - 1);
03363
03364             double[] matrixElements = (from el in transformMatrix.Split(',') select double.Parse(el,
System.Globalization.CultureInfo.InvariantCulture)).ToArray();
03365
03366             double a = matrixElements[0];
03367             double b = matrixElements[1];
03368             double c = matrixElements[2];
03369             double d = matrixElements[3];
03370             double e = matrixElements[4];
03371             double f = matrixElements[5];
03372
03373             double delta = a * d - b * c;
03374
03375             Point translation = new Point(e, f);
03376             double rotation = 0;
03377             Size scale = new Size(0, 0);
03378             Size skew = new Size(0, 0);
03379
03380             if (a != 0 || b != 0)
03381             {
03382                 double r = Math.Sqrt(a * a + b * b);
03383                 rotation = b > 0 ?  Math.Acos(a / r) :  -Math.Acos(a / r);
03384                 scale = new Size(r, delta / r);
03385                 new Size(Math.Atan((a * c + b * d) / (r * r)), 0);
03386             }
03387             else if (c != 0 || d != 0)
03388             {
03389                 double s = Math.Sqrt(c * c + d * d);
03390                 rotation = Math.PI / 2 - (d > 0 ?  Math.Acos(-c / s) :  -Math.Acos(c / s));
03391                 scale = new Size(delta / s, s);
03392                 skew = new Size(0, Math.Atan((a * c + b * d) / (s * s)));
03393             }
03394             else
03395             {
03396                 // a = b = c = d = 0
03397             }
```

```
03398
03399             return (translation, rotation, scale, skew);
03400         }
03401
03402         private static Dictionary<string, XmlElement> GetTaggedElements(XmlNode node)
03403         {
03404             Dictionary<string, XmlElement> tbr = new Dictionary<string, XmlElement>();
03405
03406             foreach (XmlNode childNode in node)
03407             {
03408                 if (childNode is XmlElement child)
03409                 {
03410                     string id = child.GetAttribute("id");
03411                     if (!string.IsNullOrEmpty(id))
03412                     {
03413                         tbr[id] = child;
03414                     }
03415
03416                     foreach (KeyValuePair<string, XmlElement> kvp in GetTaggedElements(child))
03417                     {
03418                         tbr[kvp.Key] = kvp.Value;
03419                     }
03420                 }
03421             }
03422
03423             return tbr;
03424         }
03425
03426         internal static XmlElement ToLinearGradient(this LinearGradientBrush brush, XmlDocument
      document, string gradientId)
03427         {
03428             XmlElement gradient = document.CreateElement("linearGradient", SVGContext.SVGNamespace);
03429
03430             gradient.SetAttribute("id", gradientId);
03431
03432             gradient.SetAttribute("gradientUnits", "userSpaceOnUse");
03433
03434             gradient.SetAttribute("x1",
      brush.StartPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03435             gradient.SetAttribute("y1",
      brush.StartPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03436             gradient.SetAttribute("x2",
      brush.EndPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03437             gradient.SetAttribute("y2",
      brush.EndPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03438
03439             int index = 0;
03440             foreach (GradientStop stop in brush.GradientStops)
03441             {
03442                 XmlElement gradientStop = document.CreateElement("stop", SVGContext.SVGNamespace);
03443
03444                 gradientStop.SetAttribute("offset",
      stop.Offset.ToString(System.Globalization.CultureInfo.InvariantCulture));
03445                 gradientStop.SetAttribute("stop-color", stop.Colour.ToCSSString(false));
03446                 gradientStop.SetAttribute("stop-opacity",
      stop.Colour.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
03447                 gradientStop.SetAttribute("id", gradientId + "/stop" + index.ToString());
03448
03449                 gradient.AppendChild(gradientStop);
03450                 index++;
03451             }
03452
03453             return gradient;
03454         }
03455
03456         internal static XmlElement ToRadialGradient(this RadialGradientBrush brush, XmlDocument
      document, string gradientId)
03457         {
03458             XmlElement gradient = document.CreateElement("radialGradient", SVGContext.SVGNamespace);
03459
03460             gradient.SetAttribute("id", gradientId);
03461
03462             gradient.SetAttribute("gradientUnits", "userSpaceOnUse");
03463
03464             gradient.SetAttribute("cx",
      brush.Centre.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03465             gradient.SetAttribute("cy",
      brush.Centre.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03466             gradient.SetAttribute("r",
      brush.Radius.ToString(System.Globalization.CultureInfo.InvariantCulture));
03467             gradient.SetAttribute("fx",
      brush.FocalPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03468             gradient.SetAttribute("fy",
      brush.FocalPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03469
03470             int index = 0;
03471             foreach (GradientStop stop in brush.GradientStops)
```

```
03472             {
03473                 XmlElement gradientStop = document.CreateElement("stop", SVGContext.SVGNamespace);
03474 
03475                 gradientStop.SetAttribute("offset",
      stop.Offset.ToString(System.Globalization.CultureInfo.InvariantCulture));
03476                 gradientStop.SetAttribute("stop-color", stop.Colour.ToCSSString(false));
03477                 gradientStop.SetAttribute("stop-opacity",
      stop.Colour.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
03478                 gradientStop.SetAttribute("id", gradientId + "/stop" + index.ToString());
03479 
03480                 gradient.AppendChild(gradientStop);
03481                 index++;
03482             }
03483 
03484             return gradient;
03485         }
03486 
03487         // Adapted from
      http://www.ericwhite.com/blog/2011/05/09/custom-formatting-of-xml-using-linq-to-xml-2/
03488         private static void WriteStartElement(XmlWriter writer, XmlElement e)
03489         {
03490             writer.WriteStartElement(e.Prefix, e.LocalName, e.NamespaceURI);
03491 
03492             foreach (XmlAttribute a in e.Attributes)
03493             {
03494                 writer.WriteAttributeString(a.Prefix, a.LocalName, a.NamespaceURI, a.Value);
03495             }
03496         }
03497 
03498         private static void WriteElement(XmlWriter writer, XmlElement e)
03499         {
03500             if (e.Name == "text")
03501             {
03502                 XmlWriterSettings settings = new XmlWriterSettings();
03503                 settings.Indent = false;
03504                 settings.OmitXmlDeclaration = true;
03505                 settings.ConformanceLevel = ConformanceLevel.Fragment;
03506                 settings.NamespaceHandling = NamespaceHandling.OmitDuplicates;
03507 
03508                 WriteStartElement(writer, e);
03509 
03510                 StringBuilder sb = new StringBuilder();
03511 
03512                 using (XmlWriter newWriter = XmlWriter.Create(sb, settings))
03513                 {
03514                     foreach (XmlNode n in e.ChildNodes)
03515                     {
03516                         n.WriteTo(newWriter);
03517                     }
03518                 }
03519 
03520                 writer.WriteRaw(sb.ToString().Replace(" xmlns=\"http://www.w3.org/2000/svg\">", ">"));
03521 
03522                 writer.WriteEndElement();
03523             }
03524             else
03525             {
03526                 WriteStartElement(writer, e);
03527                 foreach (XmlNode n in e.ChildNodes)
03528                 {
03529                     if (n is XmlElement element)
03530                     {
03531                         WriteElement(writer, element);
03532                     }
03533                     else
03534                     {
03535                         n.WriteTo(writer);
03536                     }
03537                 }
03538                 writer.WriteEndElement();
03539             }
03540         }
03541 
03542         private static void WriteXMLToStream(XmlElement element, Stream output)
03543         {
03544             XmlWriterSettings settings = new XmlWriterSettings();
03545             settings.Indent = true;
03546 
03547             using (XmlWriter writer = XmlWriter.Create(output, settings))
03548             {
03549                 WriteElement(writer, element);
03550             }
03551         }
03552     }
03553 }
```

8.45 SVGParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using ExCSS;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022 using System.Linq;
00023 using System.Net.Sockets;
00024 using System.Text;
00025 using System.Text.RegularExpressions;
00026 using System.Xml;
00027 using VectSharp.Filters;
00028 using static System.Net.Mime.MediaTypeNames;
00029
00030 namespace VectSharp.SVG
00031 {
00032 /// <summary>
00033 /// Contains methods to read an SVG image file.
00034 /// </summary>
00035     public static class Parser
00036     {
00037         static Parser()
00038         {
00039             ParseImageURI = ParseSVGURI;
00040         }
00041
00042 /// <summary>
00043 /// A function that takes as input an image URI and a boolean value indicating whether the image
      should be interpolated, and returns a <see cref="Page"/> object containing the image.
00044 /// By default, this is equal to <see cref="ParseSVGURI"/>, i.e.  it is only able to parse SVG images.
      If you wish to enable the parsing of other formats, you should install the "VectSharp.MuPDFUtils"
      NuGet package
00045 /// and enable the parser in your program by doing something like:
00046 /// <code>VectSharp.SVG.Parser.ParseImageURI =
      VectSharp.MuPDFUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00047 /// </summary>
00048         public static Func<string, bool, Page> ParseImageURI;
00049
00050 /// <summary>
00051 /// Parses an SVG image URI.
00052 /// </summary>
00053 /// <param name="uri">The image URI to parse.</param>
00054 /// <param name="ignored">This value is ignored and is only needed for compatibility.</param>
00055 /// <returns>A <see cref="Page"/> containing the parsed SVG image, or null.</returns>
00056         public static Page ParseSVGURI(string uri, bool ignored = false)
00057         {
00058             if (uri.StartsWith("data:"))
00059             {
00060                 string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
      uri.IndexOf(":") - 1);
00061
00062                 string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") - uri.IndexOf(";")
      - 1);
00063
00064                 if (mimeType == "image/svg+xml")
00065                 {
00066                     int offset = uri.IndexOf(",") + 1;
00067
00068                     string data;
00069
00070                     switch (type)
00071                     {
00072                         case "base64":
00073                             data =
      Encoding.UTF8.GetString(Convert.FromBase64String(uri.Substring(offset)));
00074                             break;
00075                         case "":
00076                         case "charset=utf-8":
00077                         case "utf-8":
00078                         case "utf8":
```

```
00079                           data = System.Web.HttpUtility.UrlDecode(uri.Substring(offset));
00080                           break;
00081                       case "charset=ascii":
00082                       case "ascii":
00083                           data = System.Web.HttpUtility.UrlDecode(uri.Substring(offset));
00084                           break;
00085                       default:
00086                           throw new InvalidDataException("Unknown data stream type!");
00087                   }
00088
00089                   try
00090                   {
00091                       StringReader sr = new StringReader(data);
00092                       string firstLine = sr.ReadLine();
00093                       sr.Dispose();
00094
00095                       if (firstLine.StartsWith("<?xml") && firstLine.EndsWith("?>"))
00096                       {
00097                           data = data.Substring(firstLine.Length + 1);
00098                       }
00099                   }
00100                   catch { }
00101
00102                   return FromString(data);
00103               }
00104               else
00105               {
00106                   return null;
00107               }
00108           }
00109
00110           return null;
00111       }
00112
00113 /// <summary>
00114 /// Parses SVG source into a <see cref="Page"/> containing the image represented by the code.
00115 /// </summary>
00116 /// <param name="svgSource">The SVG source code.</param>
00117 /// <returns>A <see cref="Page"/> containing the image represented by the <paramref
      name="svgSource"/>.</returns>
00118       public static Page FromString(string svgSource)
00119       {
00120           XmlDocument svgDoc = new XmlDocument();
00121           svgDoc.LoadXml(svgSource);
00122
00123           Dictionary<string, FontFamily> embeddedFonts = new Dictionary<string, FontFamily>();
00124
00125           StylesheetParser parser = new StylesheetParser();
00126
00127           List<Stylesheet> styleSheets = new List<Stylesheet>();
00128
00129           foreach (XmlNode styleNode in svgDoc.GetElementsByTagName("style"))
00130           {
00131               foreach (KeyValuePair<string, FontFamily> fnt in
      GetEmbeddedFonts(styleNode.InnerText))
00132               {
00133                   embeddedFonts.Add(fnt.Key, fnt.Value);
00134               }
00135
00136               try
00137               {
00138                   Stylesheet sheet = parser.Parse(styleNode.InnerText);
00139                   styleSheets.Add(sheet);
00140               }
00141               catch { }
00142           }
00143
00144           Dictionary<string, (bool, Brush)> gradients = new Dictionary<string, (bool, Brush)>();
00145           Dictionary<string, IFilter> filters = new Dictionary<string, IFilter>();
00146           Dictionary<string, XmlNode> masks = new Dictionary<string, XmlNode>();
00147
00148
00149           foreach (KeyValuePair<string, (bool, Brush)> fnt in
      GetGradients(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00150           {
00151               gradients.Add(fnt.Key, fnt.Value);
00152           }
00153
00154           foreach (KeyValuePair<string, IFilter> filt in
      GetFilters(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00155           {
00156               filters.Add(filt.Key, filt.Value);
00157           }
00158
00159           foreach (KeyValuePair<string, XmlNode> mask in
      GetMasks(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00160           {
```

```
00161                 masks.Add(mask.Key, mask.Value);
00162             }
00163 
00164             foreach (XmlNode definitionsNode in svgDoc.GetElementsByTagName("defs"))
00165             {
00166                 foreach (KeyValuePair<string, (bool, Brush)> fnt in GetGradients(definitionsNode,
      styleSheets))
00167                 {
00168                     gradients.Add(fnt.Key, fnt.Value);
00169                 }
00170 
00171                 foreach (KeyValuePair<string, IFilter> filt in GetFilters(definitionsNode,
      styleSheets))
00172                 {
00173                     filters.Add(filt.Key, filt.Value);
00174                 }
00175 
00176                 foreach (KeyValuePair<string, XmlNode> mask in GetMasks(definitionsNode, styleSheets))
00177                 {
00178                     masks.Add(mask.Key, mask.Value);
00179                 }
00180             }
00181 
00182 
00183 
00184             Graphics gpr = new Graphics();
00185 
00186             Size pageSize = InterpretSVGObject(svgDoc.GetElementsByTagName("svg")[0], gpr, new
      PresentationAttributes() { EmbeddedFonts = embeddedFonts }, styleSheets, gradients, filters, masks);
00187 
00188             Page pg = new Page(pageSize.Width, pageSize.Height);
00189 
00190             pg.Graphics = gpr;
00191 
00192             return pg;
00193         }
00194 
00195 /// <summary>
00196 /// Parses an SVG image file into a <see cref="Page"/> containing the image.
00197 /// </summary>
00198 /// <param name="fileName">The path to the SVG image file.</param>
00199 /// <returns>A <see cref="Page"/> containing the image represented by the file.</returns>
00200         public static Page FromFile(string fileName)
00201         {
00202             return FromString(File.ReadAllText(fileName));
00203         }
00204 
00205 /// <summary>
00206 /// Parses an stream containing SVG source code into a <see cref="Page"/> containing the image
      represented by the code.
00207 /// </summary>
00208 /// <param name="svgSourceStream">The stream containing SVG source code.</param>
00209 /// <returns>A <see cref="Page"/> containing the image represented by the <paramref
      name="svgSourceStream"/>.</returns>
00210         public static Page FromStream(Stream svgSourceStream)
00211         {
00212             using (StreamReader sr = new StreamReader(svgSourceStream))
00213             {
00214                 return FromString(sr.ReadToEnd());
00215             }
00216         }
00217 
00218         private static Size InterpretSVGObject(XmlNode svgObject, Graphics gpr, PresentationAttributes
      attributes, IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients,
      Dictionary<string, IFilter> filters, Dictionary<string, XmlNode> masks)
00219         {
00220             double[] viewBox = ParseListOfDoubles(svgObject.Attributes?["viewBox"]?.Value);
00221 
00222             double width, height, x, y;
00223 
00224             string widthAttribute = svgObject.Attributes?["width"]?.Value?.Replace("px",
      "")?.Replace("pt", "")?.Replace("mm", "")?.Replace("in", "")?.Replace("cm", "");
00225 
00226             if (!double.TryParse(widthAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out width)) { width = double.NaN; }
00227 
00228             string heightAttribute = svgObject.Attributes?["height"]?.Value?.Replace("px",
      "")?.Replace("pt", "")?.Replace("mm", "")?.Replace("in", "")?.Replace("cm", "");
00229             if (!double.TryParse(heightAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out height)) { height = double.NaN; }
00230 
00231             string xAttribute = svgObject.Attributes?["x"]?.Value;
00232             double.TryParse(xAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out x);
00233 
00234             string yAttribute = svgObject.Attributes?["y"]?.Value;
00235             double.TryParse(yAttribute, System.Globalization.NumberStyles.Any,
```

```
      System.Globalization.CultureInfo.InvariantCulture, out y);
00236
00237             double scaleX = 1;
00238             double scaleY = 1;
00239
00240             double postTranslateX = 0;
00241             double postTranslateY = 0;
00242
00243             if (viewBox != null)
00244             {
00245                 if (!double.IsNaN(width) && !double.IsNaN(height))
00246                 {
00247                     scaleX = width / viewBox[2];
00248                     scaleY = height / viewBox[3];
00249                 }
00250                 else if (!double.IsNaN(width) && double.IsNaN(height))
00251                 {
00252                     scaleX = width / viewBox[2];
00253                     scaleY = scaleX;
00254                     height = scaleY * viewBox[3];
00255                 }
00256                 else if (double.IsNaN(width) && !double.IsNaN(height))
00257                 {
00258                     scaleY = height / viewBox[3];
00259                     scaleX = scaleY;
00260                     width = scaleX * viewBox[2];
00261                 }
00262                 else if (double.IsNaN(width) && double.IsNaN(height))
00263                 {
00264                     width = viewBox[2];
00265                     height = viewBox[3];
00266                 }
00267
00268                 postTranslateX = -viewBox[0];
00269                 postTranslateY = -viewBox[1];
00270             }
00271             else
00272             {
00273                 viewBox = new double[4];
00274
00275                 if (!double.IsNaN(width))
00276                 {
00277                     viewBox[2] = width;
00278                 }
00279
00280                 if (!double.IsNaN(height))
00281                 {
00282                     viewBox[3] = height;
00283                 }
00284             }
00285
00286             double diagonal = Math.Sqrt(viewBox[2] * viewBox[2] + viewBox[3] * viewBox[3]) /
      Math.Sqrt(2);
00287
00288             Size tbrSize = new Size(width, height);
00289
00290             gpr.Save();
00291             gpr.Translate(x, y);
00292             gpr.Scale(scaleX, scaleY);
00293             gpr.Translate(postTranslateX, postTranslateY);
00294
00295             attributes = InterpretPresentationAttributes(svgObject, attributes, viewBox[2],
      viewBox[3], diagonal, gpr, styleSheets, gradients);
00296
00297             foreach (KeyValuePair<string, XmlNode> mask in masks)
00298             {
00299                 Graphics maskGpr = new Graphics();
00300                 InterpretGObject(mask.Value, maskGpr, viewBox[2], viewBox[3], diagonal, attributes,
      styleSheets, gradients, filters);
00301
00302                 filters.Add(mask.Key, new MaskFilter(maskGpr));
00303             }
00304
00305             InterpretSVGChildren(svgObject, gpr, attributes, viewBox[2], viewBox[3], diagonal,
      styleSheets, gradients, filters);
00306
00307             gpr.Restore();
00308
00309             return tbrSize;
00310         }
00311
00312         private static void InterpretSVGChildren(XmlNode svgObject, Graphics gpr,
      PresentationAttributes attributes, double width, double height, double diagonal,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients, Dictionary<string,
      IFilter> filters)
00313         {
00314             foreach (XmlNode child in svgObject.ChildNodes)
```

```
00315             {
00316                 InterpretSVGElement(child, gpr, attributes, width, height, diagonal, styleSheets,
      gradients, filters);
00317             }
00318         }
00319 
00320         private static void InterpretSVGElement(XmlNode currObject, Graphics gpr,
      PresentationAttributes attributes, double width, double height, double diagonal,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients, Dictionary<string,
      IFilter> filters)
00321         {
00322             if (currObject.NodeType == XmlNodeType.EntityReference)
00323             {
00324                 InterpretSVGChildren(currObject, gpr, attributes, width, height, diagonal,
      styleSheets, gradients, filters);
00325             }
00326             else if (currObject.Name.Equals("svg", StringComparison.OrdinalIgnoreCase))
00327             {
00328                 InterpretSVGObject(currObject, gpr, attributes, styleSheets, gradients, filters, new
      Dictionary<string, XmlNode>());
00329             }
00330             else if (currObject.Name.Equals("line", StringComparison.OrdinalIgnoreCase))
00331             {
00332                 InterpretLineObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00333             }
00334             else if (currObject.Name.Equals("circle", StringComparison.OrdinalIgnoreCase))
00335             {
00336                 InterpretCircleObject(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00337             }
00338             else if (currObject.Name.Equals("ellipse", StringComparison.OrdinalIgnoreCase))
00339             {
00340                 InterpretEllipseObject(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00341             }
00342             else if (currObject.Name.Equals("path", StringComparison.OrdinalIgnoreCase))
00343             {
00344                 InterpretPathObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00345             }
00346             else if (currObject.Name.Equals("polyline", StringComparison.OrdinalIgnoreCase))
00347             {
00348                 InterpretPolyLineObject(currObject, false, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00349             }
00350             else if (currObject.Name.Equals("polygon", StringComparison.OrdinalIgnoreCase))
00351             {
00352                 InterpretPolyLineObject(currObject, true, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00353             }
00354             else if (currObject.Name.Equals("rect", StringComparison.OrdinalIgnoreCase))
00355             {
00356                 InterpretRectObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00357             }
00358             else if (currObject.Name.Equals("use", StringComparison.OrdinalIgnoreCase))
00359             {
00360                 InterpretUseObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients, filters);
00361             }
00362             else if (currObject.Name.Equals("g", StringComparison.OrdinalIgnoreCase) ||
      currObject.Name.Equals("symbol", StringComparison.OrdinalIgnoreCase))
00363             {
00364                 InterpretGObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients, filters);
00365             }
00366             else if (currObject.Name.Equals("text", StringComparison.OrdinalIgnoreCase))
00367             {
00368                 double x = 0;
00369                 double y = 0;
00370 
00371                 InterpretTextObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients, ref x, ref y);
00372             }
00373             else if (currObject.Name.Equals("image", StringComparison.OrdinalIgnoreCase))
00374             {
00375                 InterpretImageObject(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00376             }
00377         }
00378 
00379         private static void InterpretImageObject(XmlNode currObject, Graphics gpr, double width,
      double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
      styleSheets, Dictionary<string, (bool, Brush)> gradients)
00380         {
00381             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
```

```
attributes, width, height, diagonal, gpr, styleSheets, gradients);
00382
00383             double x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width,
currAttributes.X);
00384             double y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height,
currAttributes.Y);
00385
00386             double w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width,
currAttributes.Width);
00387             double h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height,
currAttributes.Height);
00388
00389             bool interpolate = !(currObject.Attributes?["image-rendering"]?.Value == "pixelated" ||
currObject.Attributes?["image-rendering"]?.Value == "optimizeSpeed");
00390
00391             string href = currObject.Attributes?["href"]?.Value;
00392
00393             if (string.IsNullOrEmpty(href))
00394             {
00395                 href = currObject.Attributes?["xlink:href"]?.Value;
00396             }
00397
00398             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
styleSheets, gradients);
00399
00400             string tag = currObject.Attributes?["id"]?.Value;
00401
00402             if (!string.IsNullOrEmpty(href) && w > 0 && h > 0)
00403             {
00404                 Page image = ParseImageURI(href, interpolate);
00405
00406                 if (image != null)
00407                 {
00408                     gpr.Save();
00409
00410                     double scaleX = w / image.Width;
00411                     double scaleY = h / image.Height;
00412
00413                     gpr.Scale(scaleX, scaleY);
00414
00415                     gpr.DrawGraphics(x / scaleX, y / scaleY, image.Graphics, tag:  tag);
00416
00417                     gpr.Restore();
00418                 }
00419                 else
00420                 {
00421                     gpr.StrokeRectangle(x, y, w, h, Colours.Red, 0.1, tag:  tag);
00422                     gpr.StrokePath(new GraphicsPath().MoveTo(x, y).LineTo(x + w, y + h).MoveTo(x + w,
y).LineTo(x, y + h), Colours.Red, 0.1, tag:  tag);
00423                 }
00424             }
00425
00426             if (hadClippingPath)
00427             {
00428                 gpr.Restore();
00429             }
00430
00431             if (currAttributes.NeedsRestore)
00432             {
00433                 gpr.Restore();
00434             }
00435         }
00436
00437         private static string GetFirstAttributeValueIncludingAncestors(XmlNode currObject, string
attribute)
00438         {
00439             string tbr = currObject.Attributes?[attribute]?.Value;
00440
00441             if (tbr != null)
00442             {
00443                 return tbr;
00444             }
00445             else
00446             {
00447                 if (currObject.ParentNode != null && !currObject.ParentNode.Name.Equals("svg",
StringComparison.OrdinalIgnoreCase))
00448                 {
00449                     return GetFirstAttributeValueIncludingAncestors(currObject.ParentNode, attribute);
00450                 }
00451                 else
00452                 {
00453                     return null;
00454                 }
00455             }
00456         }
00457
00458         private static void InterpretTextObject(XmlNode currObject, Graphics gpr, double width, double
```

```
    height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
    Dictionary<string, (bool, Brush)> gradients, ref double x, ref double y, double fontSize = double.NaN,
    string fontFamily = null, string textAlign = null)
00459         {
00460             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
    attributes, width, height, diagonal, gpr, styleSheets, gradients);
00461
00462             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width, x);
00463             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height, y);
00464
00465             double dx = ParseLengthOrPercentage(currObject.Attributes?["dx"]?.Value, width, 0);
00466             double dy = ParseLengthOrPercentage(currObject.Attributes?["dy"]?.Value, height, 0);
00467
00468             x += dx;
00469             y += dy;
00470
00471             fontFamily = GetFirstAttributeValueIncludingAncestors(currObject, "font-family") ??
    fontFamily;
00472             fontSize = ParseLengthOrPercentage(GetFirstAttributeValueIncludingAncestors(currObject,
    "font-size"), width, fontSize);
00473             textAlign = GetFirstAttributeValueIncludingAncestors(currObject, "text-align") ??
    GetFirstAttributeValueIncludingAncestors(currObject, "text-anchor") ??  textAlign;
00474             if (textAlign == "start")
00475             {
00476                 textAlign = "left";
00477             }
00478             else if (textAlign == "middle")
00479             {
00480                 textAlign = "center";
00481             }
00482             else if (textAlign == "end")
00483             {
00484                 textAlign = "right";
00485             }
00486
00487             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
    styleSheets, gradients);
00488
00489             string tag = currObject.Attributes?["id"]?.Value;
00490
00491             if (currObject.ChildNodes.OfType<XmlNode>().Any(a => a.NodeType != XmlNodeType.Text))
00492             {
00493                 foreach (XmlNode child in currObject.ChildNodes)
00494                 {
00495                     InterpretTextObject(child, gpr, width, height, diagonal, currAttributes,
    styleSheets, gradients, ref x, ref y, fontSize, fontFamily, textAlign);
00496                 }
00497
00498                 if (hadClippingPath)
00499                 {
00500                     gpr.Restore();
00501                 }
00502
00503                 if (currAttributes.NeedsRestore)
00504                 {
00505                     gpr.Restore();
00506                 }
00507             }
00508             else
00509             {
00510                 string text = currObject.InnerText;
00511
00512                 if (!double.IsNaN(fontSize) && !string.IsNullOrEmpty(text))
00513                 {
00514                     text = text.Replace("\u00A0", " ");
00515
00516                     FontFamily parsedFontFamily = ParseFontFamily(fontFamily,
    currAttributes.EmbeddedFonts);
00517                     string fontWeight = GetFirstAttributeValueIncludingAncestors(currObject,
    "font-weight");
00518                     string fontStyle = GetFirstAttributeValueIncludingAncestors(currObject,
    "font-style");
00519
00520                     bool isBold = false;
00521                     bool isItalic = false;
00522
00523                     if (fontWeight != null && (fontWeight.Equals("bold",
    StringComparison.OrdinalIgnoreCase) || fontWeight.Equals("bolder", StringComparison.OrdinalIgnoreCase)
    || (int.TryParse(fontWeight, out int weight) && weight >= 500)))
00524                     {
00525                         isBold = true;
00526                     }
00527
00528                     if (fontStyle != null && (fontStyle.Equals("italic",
    StringComparison.OrdinalIgnoreCase) || fontStyle.Equals("oblique",
    StringComparison.OrdinalIgnoreCase)))
00529                     {
```

```
00530                       isItalic = true;
00531                   }
00532
00533                   if (isBold && !isItalic)
00534                   {
00535                       parsedFontFamily = GetBoldFontFamily(parsedFontFamily);
00536                   }
00537                   else if (isItalic && !isBold)
00538                   {
00539                       parsedFontFamily = GetItalicFontFamily(parsedFontFamily);
00540                   }
00541                   else if (isItalic && isBold)
00542                   {
00543                       parsedFontFamily = GetBoldItalicFontFamily(parsedFontFamily);
00544                   }
00545
00546
00547                   Font fnt = new Font(parsedFontFamily, fontSize);
00548
00549                   double endX = x;
00550
00551                   if (fnt.FontFamily.TrueTypeFile != null)
00552                   {
00553                       Font.DetailedFontMetrics metrics = fnt.MeasureTextAdvanced(text);
00554                       x += metrics.LeftSideBearing;
00555
00556                       if (!string.IsNullOrEmpty(textAlign) && (textAlign.Equals("right",
      StringComparison.OrdinalIgnoreCase) || textAlign.Equals("end", StringComparison.OrdinalIgnoreCase)))
00557                       {
00558                           x -= metrics.AdvanceWidth;
00559                       }
00560                       else if (!string.IsNullOrEmpty(textAlign) && textAlign.Equals("center",
      StringComparison.OrdinalIgnoreCase))
00561                       {
00562                           x -= metrics.AdvanceWidth * 0.5;
00563                       }
00564
00565                       endX += metrics.AdvanceWidth;
00566                   }
00567
00568                   TextBaselines baseline = TextBaselines.Baseline;
00569
00570                   string textBaseline = GetFirstAttributeValueIncludingAncestors(currObject,
      "alignment-baseline");
00571
00572                   if (textBaseline != null)
00573                   {
00574                       if (textBaseline.Equals("text-bottom", StringComparison.OrdinalIgnoreCase) ||
      textBaseline.Equals("bottom", StringComparison.OrdinalIgnoreCase))
00575                       {
00576                           baseline = TextBaselines.Bottom;
00577                       }
00578                       if (textBaseline.Equals("middle", StringComparison.OrdinalIgnoreCase) ||
      textBaseline.Equals("central", StringComparison.OrdinalIgnoreCase) || textBaseline.Equals("center",
      StringComparison.OrdinalIgnoreCase))
00579                       {
00580                           baseline = TextBaselines.Middle;
00581                       }
00582                       if (textBaseline.Equals("text-top", StringComparison.OrdinalIgnoreCase) ||
      textBaseline.Equals("top", StringComparison.OrdinalIgnoreCase) || textBaseline.Equals("hanging",
      StringComparison.OrdinalIgnoreCase))
00583                       {
00584                           baseline = TextBaselines.Top;
00585                       }
00586                   }
00587
00588                   if (currAttributes.StrokeFirst)
00589                   {
00590                       if (currAttributes.Stroke != null)
00591                       {
00592                           Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
00593
00594                           double[,] transform = null;
00595
00596                           if (currAttributes.StrokeGradientNeedsTransform)
00597                           {
00598                               (strokeColour, transform) = TransformBrush(strokeColour, new
      GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00599                           }
00600
00601                           if (transform != null)
00602                           {
00603                               GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
      baseline);
00604                               pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00605
```

```
00606                                 gpr.Save();
00607                                 gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00608                                 gpr.StrokePath(pth, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
00609                                 gpr.Restore();
00610                             }
00611                             else
00612                             {
00613                                 gpr.StrokeText(x, y, text, fnt, strokeColour, baseline,
      currAttributes.StrokeThickness, currAttributes.LineCap, currAttributes.LineJoin,
      currAttributes.LineDash, tag:  tag);
00614                             }
00615                         }
00616
00617                         if (currAttributes.Fill != null)
00618                         {
00619                             Brush fillColour =
      currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity * currAttributes.FillOpacity);
00620
00621                             double[,] transform = null;
00622
00623                             if (currAttributes.FillGradientNeedsTransform)
00624                             {
00625                                 (fillColour, transform) = TransformBrush(fillColour, new
      GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00626                             }
00627
00628                             if (transform != null)
00629                             {
00630                                 GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
      baseline);
00631                                 pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00632
00633                                 gpr.Save();
00634                                 gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00635                                 gpr.FillPath(pth, fillColour, tag:  tag);
00636                                 gpr.Restore();
00637                             }
00638                             else
00639                             {
00640                                 gpr.FillText(x, y, text, fnt, fillColour, baseline, tag:  tag);
00641                             }
00642                         }
00643                     }
00644                     else
00645                     {
00646                         if (currAttributes.Fill != null)
00647                         {
00648                             Brush fillColour =
      currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity * currAttributes.FillOpacity);
00649
00650                             double[,] transform = null;
00651
00652                             if (currAttributes.FillGradientNeedsTransform)
00653                             {
00654                                 (fillColour, transform) = TransformBrush(fillColour, new
      GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00655                             }
00656
00657                             if (transform != null)
00658                             {
00659                                 GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
      baseline);
00660                                 pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00661
00662                                 gpr.Save();
00663                                 gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00664                                 gpr.FillPath(pth, fillColour, tag:  tag);
00665                                 gpr.Restore();
00666                             }
00667                             else
00668                             {
00669                                 gpr.FillText(x, y, text, fnt, fillColour, baseline, tag:  tag);
00670                             }
00671                         }
00672
00673                         if (currAttributes.Stroke != null)
00674                         {
00675                             Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
00676                             double[,] transform = null;
00677
00678                             if (currAttributes.StrokeGradientNeedsTransform)
00679                             {
```

```
00680                                   (strokeColour, transform) = TransformBrush(strokeColour, new
      GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00681                               }
00682
00683                               if (transform != null)
00684                               {
00685                                   GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
      baseline);
00686                                   pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00687
00688                                   gpr.Save();
00689                                   gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00690                                   gpr.StrokePath(pth, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
00691                                   gpr.Restore();
00692                               }
00693                               else
00694                               {
00695                                   gpr.StrokeText(x, y, text, fnt, strokeColour, baseline,
      currAttributes.StrokeThickness, currAttributes.LineCap, currAttributes.LineJoin,
      currAttributes.LineDash, tag:  tag);
00696                               }
00697                           }
00698                       }
00699
00700                       x = endX;
00701                   }
00702
00703                   if (hadClippingPath)
00704                   {
00705                       gpr.Restore();
00706                   }
00707
00708                   if (currAttributes.NeedsRestore)
00709                   {
00710                       gpr.Restore();
00711                   }
00712               }
00713           }
00714
00715           private static (Brush, double[,]) TransformBrush(Brush brush, Rectangle boundingBox)
00716           {
00717               if (brush is SolidColourBrush)
00718               {
00719                   return (brush, null);
00720               }
00721               else if (brush is LinearGradientBrush linear)
00722               {
00723                   double[,] transformMatrix = MatrixUtils.Identity;
00724                   transformMatrix = MatrixUtils.Translate(transformMatrix, boundingBox.Location.X,
      boundingBox.Location.Y);
00725                   transformMatrix = MatrixUtils.Scale(transformMatrix, boundingBox.Size.Width,
      boundingBox.Size.Height);
00726
00727                   //LinearGradientBrush tbr = new
      LinearGradientBrush(MatrixUtils.Multiply(transformMatrix, linear.StartPoint),
      MatrixUtils.Multiply(transformMatrix, linear.EndPoint), linear.GradientStops);
00728                   return (brush, transformMatrix);
00729               }
00730               else if (brush is RadialGradientBrush radial)
00731               {
00732                   double[,] transformMatrix = MatrixUtils.Identity;
00733                   transformMatrix = MatrixUtils.Translate(transformMatrix, boundingBox.Location.X,
      boundingBox.Location.Y);
00734                   transformMatrix = MatrixUtils.Scale(transformMatrix, boundingBox.Size.Width,
      boundingBox.Size.Height);
00735
00736                   double determinant = transformMatrix[0, 0] * (transformMatrix[1, 1] *
      transformMatrix[2, 2] - transformMatrix[1, 2] * transformMatrix[2, 1]) -
00737                                        transformMatrix[0, 1] * (transformMatrix[1, 0] * transformMatrix[2, 2] -
      transformMatrix[1, 2] * transformMatrix[2, 0]) +
00738                                        transformMatrix[0, 2] * (transformMatrix[1, 0] * transformMatrix[2, 1] -
      transformMatrix[1, 1] * transformMatrix[2, 0]);
00739
00740                   //RadialGradientBrush tbr = new
      RadialGradientBrush(MatrixUtils.Multiply(transformMatrix, radial.FocalPoint),
      MatrixUtils.Multiply(transformMatrix, radial.Centre), radial.Radius * Math.Sqrt(determinant),
      radial.GradientStops);
00741
00742                   return (brush, transformMatrix);
00743               }
00744               else
00745               {
00746                   return (brush, null);
00747               }
00748
```

```
00749         }
00750
00751         private static string[] BoldPredicates = new string[] { "-Bold", "-bold", " Bold", " bold" };
00752         private static string[] ItalicPredicates = new string[] { "-Italic", "-italic", " Italic", "
      italic", "-Oblique", "-oblique", " Oblique", " oblique" };
00753         private static string[] BoldItalicPredicates = new string[] { "-BoldItalic", "-bolditalic", "
      BoldItalic", " bolditalic", " Bold Italic", " bold italic", "-BoldOblique", "-boldoblique", "
      BoldOblique", " boldoblique", " Bold Oblique", " bold oblique" };
00754
00755         private static FontFamily GetBoldFontFamily(FontFamily fontFamily)
00756         {
00757             switch (fontFamily.FileName)
00758             {
00759                 case "Times-Roman":
00760                 case "Times-Bold":
00761                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold);
00762                 case "Times-Italic":
00763                 case "Times-BoldItalic":
00764                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00765                 case "Helvetica":
00766                 case "Helvetica-Bold":
00767                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold);
00768                 case "Helvetica-Oblique":
00769                 case "Helvetica-BoldOblique":
00770                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00771                 case "Courier":
00772                 case "Courier-Bold":
00773                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold);
00774                 case "Courier-Oblique":
00775                 case "Courier-BoldOblique":
00776                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00777                 default:
00778                     foreach (string sr in BoldPredicates)
00779                     {
00780                         FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00781                         if (attempt != null && attempt.TrueTypeFile != null)
00782                         {
00783                             return attempt;
00784                         }
00785                         attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00786                         if (attempt != null && attempt.TrueTypeFile != null)
00787                         {
00788                             return attempt;
00789                         }
00790                     }
00791                     return fontFamily;
00792             }
00793         }
00794
00795         private static FontFamily GetItalicFontFamily(FontFamily fontFamily)
00796         {
00797             switch (fontFamily.FileName)
00798             {
00799                 case "Times-Roman":
00800                 case "Times-Italic":
00801                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00802                 case "Times-Bold":
00803                 case "Times-BoldItalic":
00804                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00805                 case "Helvetica":
00806                 case "Helvetica-Oblique":
00807                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00808                 case "Helvetica-Bold":
00809                 case "Helvetica-BoldOblique":
00810                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00811                 case "Courier":
00812                 case "Courier-Oblique":
00813                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00814                 case "Courier-Bold":
00815                 case "Courier-BoldOblique":
00816                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00817                 default:
00818                     foreach (string sr in ItalicPredicates)
00819                     {
00820                         FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00821                         if (attempt != null && attempt.TrueTypeFile != null)
00822                         {
00823                             return attempt;
```

```
00824                         }
00825                         attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00826                         if (attempt != null && attempt.TrueTypeFile != null)
00827                         {
00828                             return attempt;
00829                         }
00830                     }
00831                     return fontFamily;
00832                 }
00833             }
00834 
00835             private static FontFamily GetBoldItalicFontFamily(FontFamily fontFamily)
00836             {
00837                 switch (fontFamily.FileName)
00838                 {
00839                     case "Times-Roman":
00840                     case "Times-Italic":
00841                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00842                     case "Times-Bold":
00843                     case "Times-BoldItalic":
00844                         return
    FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00845                     case "Helvetica":
00846                     case "Helvetica-Oblique":
00847                         return
    FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00848                     case "Helvetica-Bold":
00849                     case "Helvetica-BoldOblique":
00850                         return
    FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00851                     case "Courier":
00852                     case "Courier-Oblique":
00853                         return
    FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00854                     case "Courier-Bold":
00855                     case "Courier-BoldOblique":
00856                         return
    FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00857                     default:
00858                         foreach (string sr in BoldItalicPredicates)
00859                         {
00860                             FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00861                             if (attempt != null && attempt.TrueTypeFile != null)
00862                             {
00863                                 return attempt;
00864                             }
00865                             attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00866                             if (attempt != null && attempt.TrueTypeFile != null)
00867                             {
00868                                 return attempt;
00869                             }
00870                         }
00871                         return fontFamily;
00872                 }
00873             }
00874 
00875             private static FontFamily ParseFontFamily(string fontFamily, Dictionary<string, FontFamily>
    embeddedFonts)
00876             {
00877                 string[] fontFamilies = Regexes.FontFamilySeparator.Split(fontFamily);
00878 
00879                 foreach (string fam in fontFamilies)
00880                 {
00881                     string family = fam.Trim().Trim(',', '"').Trim();
00882 
00883                     if (embeddedFonts.TryGetValue(family, out FontFamily tbr))
00884                     {
00885                         return tbr;
00886                     }
00887 
00888                     List<(string, int)> matchedFamilies = new List<(string, int)>();
00889 
00890                     for (int i = 0; i < FontFamily.StandardFamilies.Length; i++)
00891                     {
00892                         if (family.StartsWith(FontFamily.StandardFamilies[i]))
00893                         {
00894                             matchedFamilies.Add((FontFamily.StandardFamilies[i],
    FontFamily.StandardFamilies[i].Length));
00895                         }
00896                     }
00897 
00898                     if (matchedFamilies.Count > 0)
00899                     {
00900                         return FontFamily.ResolveFontFamily((from el in matchedFamilies orderby el.Item2
    descending select el.Item1).First());
00901                     }
00902                     else
```

```
00903                 {
00904                     if (family.Equals("serif", StringComparison.OrdinalIgnoreCase))
00905                     {
00906                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman);
00907                     }
00908                     else if (family.Equals("sans-serif", StringComparison.OrdinalIgnoreCase))
00909                     {
00910                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00911                     }
00912                     else if (family.Equals("monospace", StringComparison.OrdinalIgnoreCase))
00913                     {
00914                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00915                     }
00916                     else if (family.Equals("cursive", StringComparison.OrdinalIgnoreCase))
00917                     {
00918                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00919                     }
00920                     else if (family.Equals("system-ui", StringComparison.OrdinalIgnoreCase))
00921                     {
00922                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00923                     }
00924                     else if (family.Equals("ui-serif", StringComparison.OrdinalIgnoreCase))
00925                     {
00926                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman);
00927                     }
00928                     else if (family.Equals("ui-sans-serif", StringComparison.OrdinalIgnoreCase))
00929                     {
00930                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00931                     }
00932                     else if (family.Equals("ui-monospace", StringComparison.OrdinalIgnoreCase))
00933                     {
00934                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00935                     }
00936                     else if (family.Equals("StandardSymbolsPS", StringComparison.OrdinalIgnoreCase))
00937                     {
00938                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Symbol);
00939                     }
00940                     else if (family.Equals("D050000L", StringComparison.OrdinalIgnoreCase))
00941                     {
00942                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.ZapfDingbats);
00943                     }
00944                 }
00945 
00946                 FontFamily parsedFamily = FontFamily.ResolveFontFamily(family);
00947 
00948                 if (parsedFamily != null && parsedFamily.TrueTypeFile != null)
00949                 {
00950                     return parsedFamily;
00951                 }
00952             }
00953 
00954             return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00955         }
00956 
00957         private static void InterpretGObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients, Dictionary<string, IFilter> filters)
00958         {
00959             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
00960 
00961             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00962 
00963             string filter = currObject.Attributes?["filter"]?.Value ??
      currObject.Attributes?["mask"]?.Value;
00964 
00965             if (!string.IsNullOrEmpty(filter) && filter.StartsWith("url(#"))
00966             {
00967                 filter = filter.Substring(5, filter.Length - 6);
00968             }
00969 
00970             if (!string.IsNullOrEmpty(filter) && filters.ContainsKey(filter))
00971             {
00972                 Graphics filteredGraphics = new Graphics();
00973 
00974                 InterpretSVGChildren(currObject, filteredGraphics, currAttributes, width, height,
      diagonal, styleSheets, gradients, filters);
00975                 gpr.DrawGraphics(0, 0, filteredGraphics, filters[filter]);
00976             }
```

```
00977             else
00978             {
00979                 InterpretSVGChildren(currObject, gpr, currAttributes, width, height, diagonal,
      styleSheets, gradients, filters);
00980             }
00981
00982             if (hadClippingPath)
00983             {
00984                 gpr.Restore();
00985             }
00986
00987             if (currAttributes.NeedsRestore)
00988             {
00989                 gpr.Restore();
00990             }
00991         }
00992
00993         private static void InterpretUseObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients, Dictionary<string, IFilter> filters)
00994         {
00995             double x, y, w, h;
00996
00997             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width);
00998             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height);
00999             w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width, double.NaN);
01000             h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height, double.NaN);
01001
01002             string id = currObject.Attributes?["href"]?.Value ??
      currObject.Attributes?["xlink:href"]?.Value;
01003
01004             if (id != null && id.StartsWith("#"))
01005             {
01006                 id = id.Substring(1);
01007
01008                 XmlNode element =
      currObject.OwnerDocument.SelectSingleNode(string.Format("//*[@id='{0}']", id));
01009
01010                 if (element != null)
01011                 {
01012                     XmlNode clone = element.Clone();
01013
01014                     currObject.AppendChild(clone);
01015
01016
01017                     PresentationAttributes currAttributes =
      InterpretPresentationAttributes(currObject, attributes, width, height, diagonal, gpr, styleSheets,
      gradients);
01018
01019
01020                     gpr.Save();
01021                     gpr.Translate(x, y);
01022
01023                     ((XmlElement)clone).SetAttribute("x", "0");
01024                     ((XmlElement)clone).SetAttribute("y", "0");
01025
01026                     if (clone.Attributes?["viewBox"] != null)
01027                     {
01028                         ((XmlElement)clone).SetAttribute("width",
      w.ToString(System.Globalization.CultureInfo.InvariantCulture));
01029                         ((XmlElement)clone).SetAttribute("height",
      h.ToString(System.Globalization.CultureInfo.InvariantCulture));
01030                     }
01031
01032                     InterpretSVGElement(clone, gpr, currAttributes, width, height, diagonal,
      styleSheets, gradients, filters);
01033
01034                     gpr.Restore();
01035
01036                     if (currAttributes.NeedsRestore)
01037                     {
01038                         gpr.Restore();
01039                     }
01040                 }
01041             }
01042         }
01043
01044         private static bool ApplyClipPath(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients)
01045         {
01046             string id = currObject.Attributes?["clip-path"]?.Value;
01047
01048             if (id != null && id.StartsWith("url(#"))
01049             {
01050                 id = id.Substring(5);
01051                 id = id.Substring(0, id.Length - 1);
```

```
01052
01053                 XmlNode element =
      currObject.OwnerDocument.SelectSingleNode(string.Format("//*[@id='{0}']", id));
01054
01055                 if (element != null && element.ChildNodes.Count == 1 &&
      element.ChildNodes[0].Name.Equals("path", StringComparison.OrdinalIgnoreCase))
01056                 {
01057                     bool hasParentClipPath = ApplyClipPath(element, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01058
01059                     Graphics pathGraphics = new Graphics();
01060                     InterpretPathObject(element.ChildNodes[0], pathGraphics, width, height, diagonal,
      attributes, styleSheets, gradients);
01061
01062                     PathTransformerGraphicsContext ptgc = new PathTransformerGraphicsContext();
01063                     pathGraphics.CopyToIGraphicsContext(ptgc);
01064
01065                     if (!hasParentClipPath)
01066                     {
01067                         gpr.Save();
01068                     }
01069
01070                     gpr.SetClippingPath(ptgc.CurrentPath);
01071
01072                     return true;
01073                 }
01074                 else if (element != null && element.ChildNodes.Count == 1 &&
      element.ChildNodes[0].Name.Equals("rect", StringComparison.OrdinalIgnoreCase))
01075                 {
01076                     bool hasParentClipPath = ApplyClipPath(element, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01077
01078                     Graphics pathGraphics = new Graphics();
01079                     InterpretRectObject(element.ChildNodes[0], pathGraphics, width, height, diagonal,
      attributes, styleSheets, gradients);
01080
01081                     PathTransformerGraphicsContext ptgc = new PathTransformerGraphicsContext();
01082                     pathGraphics.CopyToIGraphicsContext(ptgc);
01083
01084                     if (!hasParentClipPath)
01085                     {
01086                         gpr.Save();
01087                     }
01088
01089                     gpr.SetClippingPath(ptgc.CurrentPath);
01090
01091                     return true;
01092                 }
01093
01094                 return false;
01095             }
01096             else
01097             {
01098                 return false;
01099             }
01100         }
01101
01102         private static void InterpretRectObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients)
01103         {
01104             double x, y, w, h, rx, ry;
01105
01106             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width);
01107             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height);
01108             w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width);
01109             h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height);
01110             rx = ParseLengthOrPercentage(currObject.Attributes?["rx"]?.Value, width, double.NaN);
01111             ry = ParseLengthOrPercentage(currObject.Attributes?["ry"]?.Value, height, double.NaN);
01112
01113             if (w > 0 && h > 0)
01114             {
01115                 if (double.IsNaN(rx) && !double.IsNaN(ry))
01116                 {
01117                     rx = ry;
01118                 }
01119                 else if (!double.IsNaN(rx) && double.IsNaN(ry))
01120                 {
01121                     ry = rx;
01122                 }
01123
01124                 if (double.IsNaN(rx))
01125                 {
01126                     rx = 0;
01127                 }
01128
01129                 if (double.IsNaN(ry))
```

```
01130                 {
01131                     ry = 0;
01132                 }
01133 
01134                 rx = Math.Min(rx, w / 2);
01135                 ry = Math.Min(ry, h / 2);
01136 
01137                 GraphicsPath path = new GraphicsPath();
01138 
01139                 path.MoveTo(x + rx, y);
01140                 path.LineTo(x + w - rx, y);
01141 
01142                 if (rx > 0 && ry > 0)
01143                 {
01144                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + w, y + ry));
01145                 }
01146 
01147                 path.LineTo(x + w, y + h - ry);
01148 
01149                 if (rx > 0 && ry > 0)
01150                 {
01151                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + w - rx, y + h));
01152                 }
01153 
01154                 path.LineTo(x + rx, y + h);
01155 
01156                 if (rx > 0 && ry > 0)
01157                 {
01158                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x, y + h - ry));
01159                 }
01160 
01161                 path.LineTo(x, y + ry);
01162 
01163                 if (rx > 0 && ry > 0)
01164                 {
01165                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + rx, y));
01166                 }
01167 
01168                 path.Close();
01169 
01170                 PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01171 
01172                 bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01173 
01174                 string tag = currObject.Attributes?["id"]?.Value;
01175 
01176                 if (currAttributes.StrokeFirst)
01177                 {
01178                     if (currAttributes.Stroke != null)
01179                     {
01180                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01181 
01182                         GraphicsPath oldPath = path;
01183                         double[,] transform = null;
01184 
01185                         if (currAttributes.StrokeGradientNeedsTransform)
01186                         {
01187                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01188                         }
01189 
01190                         if (transform != null)
01191                         {
01192                             gpr.Save();
01193                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01194                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01195                         }
01196 
01197                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01198 
01199                         if (transform != null)
01200                         {
01201                             gpr.Restore();
01202                             path = oldPath;
01203                         }
01204                     }
01205 
01206                     if (currAttributes.Fill != null)
01207                     {
01208                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01209 
```

```
01210                         GraphicsPath oldPath = path;
01211                         double[,] transform = null;
01212 
01213                         if (currAttributes.FillGradientNeedsTransform)
01214                         {
01215                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01216                         }
01217 
01218                         if (transform != null)
01219                         {
01220                             gpr.Save();
01221                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01222                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01223                         }
01224 
01225                         gpr.FillPath(path, fillColour, tag:  tag);
01226 
01227                         if (transform != null)
01228                         {
01229                             gpr.Restore();
01230                             path = oldPath;
01231                         }
01232                     }
01233                 }
01234                 else
01235                 {
01236                     if (currAttributes.Fill != null)
01237                     {
01238                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01239 
01240                         GraphicsPath oldPath = path;
01241                         double[,] transform = null;
01242 
01243                         if (currAttributes.FillGradientNeedsTransform)
01244                         {
01245                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01246                         }
01247 
01248                         if (transform != null)
01249                         {
01250                             gpr.Save();
01251                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01252                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01253                         }
01254 
01255                         gpr.FillPath(path, fillColour, tag:  tag);
01256 
01257                         if (transform != null)
01258                         {
01259                             gpr.Restore();
01260                             path = oldPath;
01261                         }
01262                     }
01263 
01264                     if (currAttributes.Stroke != null)
01265                     {
01266                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01267 
01268                         GraphicsPath oldPath = path;
01269                         double[,] transform = null;
01270 
01271                         if (currAttributes.StrokeGradientNeedsTransform)
01272                         {
01273                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01274                         }
01275 
01276                         if (transform != null)
01277                         {
01278                             gpr.Save();
01279                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01280                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01281                         }
01282 
01283                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01284 
01285                         if (transform != null)
01286                         {
01287                             gpr.Restore();
01288                             path = oldPath;
01289                         }
```

```
01290                     }
01291                 }
01292
01293                 if (hadClippingPath)
01294                 {
01295                     gpr.Restore();
01296                 }
01297
01298                 if (currAttributes.NeedsRestore)
01299                 {
01300                     gpr.Restore();
01301                 }
01302             }
01303         }
01304
01305         private static void InterpretPolyLineObject(XmlNode currObject, bool isPolygon, Graphics gpr,
      double width, double height, double diagonal, PresentationAttributes attributes,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients)
01306         {
01307             string points = currObject.Attributes?["points"]?.Value;
01308
01309             if (points != null)
01310             {
01311                 double[] coordinates = ParseListOfDoubles(points);
01312
01313                 GraphicsPath path = new GraphicsPath();
01314
01315                 for (int i = 0; i < coordinates.Length; i += 2)
01316                 {
01317                     path.LineTo(coordinates[i], coordinates[i + 1]);
01318                 }
01319
01320                 if (isPolygon)
01321                 {
01322                     path.Close();
01323                 }
01324
01325                 PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01326
01327                 bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01328
01329                 string tag = currObject.Attributes?["id"]?.Value;
01330
01331                 if (currAttributes.StrokeFirst)
01332                 {
01333                     if (currAttributes.Stroke != null)
01334                     {
01335                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01336
01337                         GraphicsPath oldPath = path;
01338                         double[,] transform = null;
01339
01340                         if (currAttributes.StrokeGradientNeedsTransform)
01341                         {
01342                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01343                         }
01344
01345                         if (transform != null)
01346                         {
01347                             gpr.Save();
01348                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01349                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01350                         }
01351
01352                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01353
01354                         if (transform != null)
01355                         {
01356                             gpr.Restore();
01357                             path = oldPath;
01358                         }
01359                     }
01360
01361                     if (currAttributes.Fill != null)
01362                     {
01363                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01364
01365                         GraphicsPath oldPath = path;
01366                         double[,] transform = null;
01367
```

```
01368                         if (currAttributes.FillGradientNeedsTransform)
01369                         {
01370                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01371                         }
01372 
01373                         if (transform != null)
01374                         {
01375                             gpr.Save();
01376                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01377                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01378                         }
01379 
01380                         gpr.FillPath(path, fillColour, tag:  tag);
01381 
01382                         if (transform != null)
01383                         {
01384                             gpr.Restore();
01385                             path = oldPath;
01386                         }
01387                     }
01388                 }
01389                 else
01390                 {
01391                     if (currAttributes.Fill != null)
01392                     {
01393                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01394 
01395                         GraphicsPath oldPath = path;
01396                         double[,] transform = null;
01397 
01398                         if (currAttributes.FillGradientNeedsTransform)
01399                         {
01400                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01401                         }
01402 
01403                         if (transform != null)
01404                         {
01405                             gpr.Save();
01406                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01407                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01408                         }
01409 
01410                         gpr.FillPath(path, fillColour, tag:  tag);
01411 
01412                         if (transform != null)
01413                         {
01414                             gpr.Restore();
01415                             path = oldPath;
01416                         }
01417                     }
01418 
01419                     if (currAttributes.Stroke != null)
01420                     {
01421                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01422 
01423                         GraphicsPath oldPath = path;
01424                         double[,] transform = null;
01425 
01426                         if (currAttributes.StrokeGradientNeedsTransform)
01427                         {
01428                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01429                         }
01430 
01431                         if (transform != null)
01432                         {
01433                             gpr.Save();
01434                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01435                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01436                         }
01437 
01438                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01439 
01440                         if (transform != null)
01441                         {
01442                             gpr.Restore();
01443                             path = oldPath;
01444                         }
01445                     }
01446                 }
01447 
```

```
01448                 if (hadClippingPath)
01449                 {
01450                     gpr.Restore();
01451                 }
01452 
01453                 if (currAttributes.NeedsRestore)
01454                 {
01455                     gpr.Restore();
01456                 }
01457             }
01458         }
01459 
01460         private static void InterpretPathObject(XmlNode currObject, Graphics gpr, double width, double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients)
01461         {
01462             string d = currObject.Attributes?["d"]?.Value;
01463 
01464             if (d != null)
01465             {
01466                 List<string> pathData = TokenisePathData(d);
01467 
01468                 GraphicsPath path = new GraphicsPath();
01469 
01470                 Point lastPoint = new Point();
01471                 Point?  figureStartPoint = null;
01472 
01473                 char lastCommand = '\0';
01474                 Point lastCtrlPoint = new Point();
01475 
01476                 for (int i = 0; i < pathData.Count; i++)
01477                 {
01478                     Point delta = new Point();
01479 
01480                     bool isAbsolute = char.IsUpper(pathData[i][0]);
01481 
01482                     if (!isAbsolute)
01483                     {
01484                         delta = lastPoint;
01485                     }
01486 
01487                     switch (pathData[i][0])
01488                     {
01489                         case 'M':
01490                         case 'm':
01491                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1], System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2], System.Globalization.CultureInfo.InvariantCulture));
01492                             path.MoveTo(lastPoint);
01493                             figureStartPoint = lastPoint;
01494                             i += 2;
01495                             lastCommand = 'M';
01496                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01497                             {
01498                                 if (!isAbsolute)
01499                                 {
01500                                     delta = lastPoint;
01501                                 }
01502                                 else
01503                                 {
01504                                     delta = new Point();
01505                                 }
01506 
01507                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1], System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2], System.Globalization.CultureInfo.InvariantCulture));
01508                                 path.LineTo(lastPoint);
01509 
01510                                 i += 2;
01511                                 lastCommand = 'L';
01512                             }
01513                             break;
01514                         case 'L':
01515                         case 'l':
01516                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1], System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2], System.Globalization.CultureInfo.InvariantCulture));
01517                             path.LineTo(lastPoint);
01518                             if (figureStartPoint == null)
01519                             {
01520                                 figureStartPoint = lastPoint;
01521                             }
01522                             i += 2;
01523                             lastCommand = 'L';
01524                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01525                             {
01526                                 if (!isAbsolute)
```

```
01527                                   {
01528                                       delta = lastPoint;
01529                                   }
01530                                   else
01531                                   {
01532                                       delta = new Point();
01533                                   }
01534
01535                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01536                                   path.LineTo(lastPoint);
01537
01538                                   i += 2;
01539                               }
01540                               break;
01541                           case 'H':
01542                           case 'h':
01543                               lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), lastPoint.Y);
01544                               path.LineTo(lastPoint);
01545                               if (figureStartPoint == null)
01546                               {
01547                                   figureStartPoint = lastPoint;
01548                               }
01549                               i++;
01550                               lastCommand = 'L';
01551                               while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01552                               {
01553                                   if (!isAbsolute)
01554                                   {
01555                                       delta = lastPoint;
01556                                   }
01557                                   else
01558                                   {
01559                                       delta = new Point();
01560                                   }
01561
01562                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), lastPoint.Y);
01563                                   path.LineTo(lastPoint);
01564
01565                                   i++;
01566                               }
01567                               break;
01568                           case 'V':
01569                           case 'v':
01570                               lastPoint = new Point(lastPoint.X, delta.Y + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture));
01571                               path.LineTo(lastPoint);
01572                               if (figureStartPoint == null)
01573                               {
01574                                   figureStartPoint = lastPoint;
01575                               }
01576                               i++;
01577                               lastCommand = 'L';
01578                               while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01579                               {
01580                                   if (!isAbsolute)
01581                                   {
01582                                       delta = lastPoint;
01583                                   }
01584                                   else
01585                                   {
01586                                       delta = new Point();
01587                                   }
01588
01589                                   lastPoint = new Point(lastPoint.X, delta.Y + double.Parse(pathData[i +
      1], System.Globalization.CultureInfo.InvariantCulture));
01590                                   path.LineTo(lastPoint);
01591
01592                                   i++;
01593                               }
01594                               break;
01595                           case 'C':
01596                           case 'c':
01597                               {
01598                                   Point ctrlPoint1 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01599                                   i += 2;
01600
01601                                   Point ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01602                                   i += 2;
01603
```

```
01604                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01605                                   i += 2;
01606
01607                                   if (figureStartPoint == null)
01608                                   {
01609                                       figureStartPoint = lastPoint;
01610                                   }
01611
01612                                   path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01613
01614                                   lastCtrlPoint = ctrlPoint2;
01615                                   lastCommand = 'C';
01616
01617                                   while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01618                                   {
01619                                       if (!isAbsolute)
01620                                       {
01621                                           delta = lastPoint;
01622                                       }
01623                                       else
01624                                       {
01625                                           delta = new Point();
01626                                       }
01627
01628                                       ctrlPoint1 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01629                                       i += 2;
01630
01631                                       ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01632                                       i += 2;
01633
01634                                       lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01635                                       i += 2;
01636
01637                                       path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01638                                       lastCtrlPoint = ctrlPoint2;
01639                                   }
01640                               }
01641                               break;
01642                           case 'S':
01643                           case 's':
01644                               {
01645                                   Point ctrlPoint1;
01646
01647                                   if (lastCommand == 'C')
01648                                   {
01649                                       ctrlPoint1 = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01650                                   }
01651                                   else
01652                                   {
01653                                       ctrlPoint1 = lastPoint;
01654                                   }
01655
01656                                   Point ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01657                                   i += 2;
01658
01659                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01660                                   i += 2;
01661
01662                                   if (figureStartPoint == null)
01663                                   {
01664                                       figureStartPoint = lastPoint;
01665                                   }
01666
01667                                   path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01668
01669                                   lastCtrlPoint = ctrlPoint2;
01670                                   lastCommand = 'C';
01671
01672                                   while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01673                                   {
01674                                       if (!isAbsolute)
01675                                       {
01676                                           delta = lastPoint;
01677                                       }
```

```
01678                                   else
01679                                   {
01680                                       delta = new Point();
01681                                   }
01682 
01683                                   ctrlPoint1 = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01684 
01685                                   ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01686                                   i += 2;
01687 
01688                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01689                                   i += 2;
01690 
01691                                   path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01692 
01693                                   lastCtrlPoint = ctrlPoint2;
01694                               }
01695                           }
01696                           break;
01697                       case 'Q':
01698                       case 'q':
01699                           {
01700                               Point ctrlPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01701                               i += 2;
01702 
01703                               Point actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01704 
01705                               lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01706                               i += 2;
01707 
01708                               Point actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01709 
01710                               if (figureStartPoint == null)
01711                               {
01712                                   figureStartPoint = lastPoint;
01713                               }
01714 
01715                               path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01716 
01717                               lastCtrlPoint = ctrlPoint;
01718                               lastCommand = 'Q';
01719 
01720                               while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01721                               {
01722                                   if (!isAbsolute)
01723                                   {
01724                                       delta = lastPoint;
01725                                   }
01726                                   else
01727                                   {
01728                                       delta = new Point();
01729                                   }
01730 
01731                                   ctrlPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01732                                   i += 2;
01733 
01734                                   actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01735 
01736                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01737                                   i += 2;
01738 
01739                                   actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01740 
01741                                   path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01742                                   lastCtrlPoint = ctrlPoint;
01743                               }
01744 
01745 
01746                           }
01747                           break;
```

```
01748                         case 'T':
01749                         case 't':
01750                             {
01751                                 Point ctrlPoint;
01752 
01753                                 if (lastCommand == 'Q')
01754                                 {
01755                                     ctrlPoint = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01756                                 }
01757                                 else
01758                                 {
01759                                     ctrlPoint = lastPoint;
01760                                 }
01761 
01762                                 Point actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01763 
01764                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01765                                 i += 2;
01766 
01767                                 Point actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01768 
01769                                 if (figureStartPoint == null)
01770                                 {
01771                                     figureStartPoint = lastPoint;
01772                                 }
01773 
01774                                 path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01775                                 lastCtrlPoint = ctrlPoint;
01776                                 lastCommand = 'Q';
01777 
01778                                 while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01779                                 {
01780                                     if (!isAbsolute)
01781                                     {
01782                                         delta = lastPoint;
01783                                     }
01784                                     else
01785                                     {
01786                                         delta = new Point();
01787                                     }
01788 
01789                                     ctrlPoint = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01790 
01791                                     actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01792 
01793                                     lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01794                                     i += 2;
01795 
01796                                     actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01797 
01798                                     path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01799 
01800                                     lastCtrlPoint = ctrlPoint;
01801                                 }
01802                             }
01803                             break;
01804                         case 'A':
01805                         case 'a':
01806                             {
01807                                 Point startPoint = lastPoint;
01808 
01809                                 if (figureStartPoint == null)
01810                                 {
01811                                     figureStartPoint = lastPoint;
01812                                 }
01813 
01814                                 Point radii = new Point(double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01815                                 double angle = double.Parse(pathData[i + 3],
      System.Globalization.CultureInfo.InvariantCulture) * Math.PI / 180;
01816                                 bool largeArcFlag = pathData[i + 4][0] == '1';
01817                                 bool sweepFlag = pathData[i + 5][0] == '1';
01818 
01819                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 6],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 7],
      System.Globalization.CultureInfo.InvariantCulture));
```

```
01820                                   i += 7;
01821
01822                                   path.EllipticalArc(radii.X, radii.Y, angle, largeArcFlag, sweepFlag,
      lastPoint);
01823
01824                                   while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01825                                   {
01826                                       if (!isAbsolute)
01827                                       {
01828                                           delta = lastPoint;
01829                                       }
01830                                       else
01831                                       {
01832                                           delta = new Point();
01833                                       }
01834
01835                                       startPoint = lastPoint;
01836                                       radii = new Point(double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01837                                       angle = double.Parse(pathData[i + 3],
      System.Globalization.CultureInfo.InvariantCulture) * Math.PI / 180;
01838                                       largeArcFlag = pathData[i + 4][0] == '1';
01839                                       sweepFlag = pathData[i + 5][0] == '1';
01840
01841                                       lastPoint = new Point(delta.X + double.Parse(pathData[i + 6],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 7],
      System.Globalization.CultureInfo.InvariantCulture));
01842                                       i += 7;
01843
01844                                       path.EllipticalArc(radii.X, radii.Y, angle, largeArcFlag,
      sweepFlag, lastPoint);
01845                                   }
01846                               }
01847
01848                               break;
01849                           case 'Z':
01850                           case 'z':
01851                               path.Close();
01852                               lastPoint = figureStartPoint.Value;
01853                               figureStartPoint = null;
01854                               lastCommand = 'Z';
01855                               break;
01856                       }
01857                   }
01858
01859                   PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01860
01861                   bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01862
01863                   string tag = currObject.Attributes?["id"]?.Value;
01864
01865                   if (currAttributes.StrokeFirst)
01866                   {
01867                       if (currAttributes.Stroke != null)
01868                       {
01869                           Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01870
01871                           GraphicsPath oldPath = path;
01872                           double[,] transform = null;
01873
01874                           if (currAttributes.StrokeGradientNeedsTransform)
01875                           {
01876                               (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01877                           }
01878
01879                           if (transform != null)
01880                           {
01881                               gpr.Save();
01882                               gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01883                               path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01884                           }
01885
01886                           gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01887
01888                           if (transform != null)
01889                           {
01890                               gpr.Restore();
01891                               path = oldPath;
01892                           }
01893                       }
```

```
01894
01895                   if (currAttributes.Fill != null)
01896                   {
01897                       Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
    * currAttributes.FillOpacity);
01898
01899                       GraphicsPath oldPath = path;
01900                       double[,] transform = null;
01901
01902                       if (currAttributes.FillGradientNeedsTransform)
01903                       {
01904                           (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01905                       }
01906
01907                       if (transform != null)
01908                       {
01909                           gpr.Save();
01910                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
01911                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01912                       }
01913
01914                       gpr.FillPath(path, fillColour, tag:  tag);
01915
01916                       if (transform != null)
01917                       {
01918                           gpr.Restore();
01919                           path = oldPath;
01920                       }
01921                   }
01922               }
01923               else
01924               {
01925                   if (currAttributes.Fill != null)
01926                   {
01927                       Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
    * currAttributes.FillOpacity);
01928
01929                       GraphicsPath oldPath = path;
01930                       double[,] transform = null;
01931
01932                       if (currAttributes.FillGradientNeedsTransform)
01933                       {
01934                           (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01935                       }
01936
01937                       if (transform != null)
01938                       {
01939                           gpr.Save();
01940                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
01941                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01942                       }
01943
01944                       gpr.FillPath(path, fillColour, tag:  tag);
01945
01946                       if (transform != null)
01947                       {
01948                           gpr.Restore();
01949                           path = oldPath;
01950                       }
01951                   }
01952
01953                   if (currAttributes.Stroke != null)
01954                   {
01955                       Brush strokeColour =
    currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01956                       GraphicsPath oldPath = path;
01957                       double[,] transform = null;
01958
01959                       if (currAttributes.StrokeGradientNeedsTransform)
01960                       {
01961                           (strokeColour, transform) = TransformBrush(strokeColour,
    path.GetBounds());
01962                       }
01963
01964                       if (transform != null)
01965                       {
01966                           gpr.Save();
01967                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
01968                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01969                       }
01970
01971                       gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
    currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01972
```

```
01973                         if (transform != null)
01974                         {
01975                             gpr.Restore();
01976                             path = oldPath;
01977                         }
01978                     }
01979                 }
01980 
01981                 if (hadClippingPath)
01982                 {
01983                     gpr.Restore();
01984                 }
01985 
01986                 if (currAttributes.NeedsRestore)
01987                 {
01988                     gpr.Restore();
01989                 }
01990             }
01991 
01992         }
01993 
01994         private static List<string> TokenisePathData(string d)
01995         {
01996             List<string> tbr = new List<string>();
01997 
01998             string currToken = "";
01999 
02000             for (int i = 0; i < d.Length; i++)
02001             {
02002                 char c = d[i];
02003 
02004                 if (c >= '0' && c <= '9' || c == 'e' || c == 'E')
02005                 {
02006                     currToken += c;
02007                 }
02008                 else if (c == '.')
02009                 {
02010                     if (!currToken.Contains('.'))
02011                     {
02012                         currToken += c;
02013                     }
02014                     else
02015                     {
02016                         if (!string.IsNullOrEmpty(currToken))
02017                         {
02018                             tbr.Add(currToken);
02019                         }
02020                         currToken = "" + c;
02021                     }
02022                 }
02023                 else if (c == '-' || c == '+')
02024                 {
02025                     if (i > 0 && (d[i - 1] == 'e' || d[i - 1] == 'E'))
02026                     {
02027                         currToken += c;
02028                     }
02029                     else
02030                     {
02031                         if (!string.IsNullOrEmpty(currToken))
02032                         {
02033                             tbr.Add(currToken);
02034                         }
02035                         currToken = "" + c;
02036                     }
02037                 }
02038                 else if (char.IsWhiteSpace(c) || c == ',')
02039                 {
02040                     if (!string.IsNullOrEmpty(currToken))
02041                     {
02042                         tbr.Add(currToken);
02043                     }
02044                     currToken = "";
02045                 }
02046                 else if (i < d.Length - 2 && (c == 'N' || c == 'n') && (d[i + 1] == 'a' || d[i + 1] ==
      'A') && (d[i + 2] == 'N' || d[i + 2] == 'n'))
02047                 {
02048                     if (!string.IsNullOrEmpty(currToken))
02049                     {
02050                         tbr.Add(currToken);
02051                     }
02052                     tbr.Add("NaN");
02053                     currToken = "";
02054                     i += 2;
02055                 }
02056                 else if ("MmLlHhVvCcSsQqTtAaZz".Contains(c))
02057                 {
02058                     if (!string.IsNullOrEmpty(currToken))
```

```
02059                     {
02060                         tbr.Add(currToken);
02061                     }
02062                     tbr.Add(c.ToString());
02063                     currToken = "";
02064                 }
02065             }
02066 
02067             if (!string.IsNullOrEmpty(currToken))
02068             {
02069                 tbr.Add(currToken);
02070             }
02071 
02072             return tbr;
02073         }
02074 
02075         private static void InterpretCircleObject(XmlNode circleObject, Graphics gpr, double width,
      double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
      styleSheets, Dictionary<string, (bool, Brush)> gradients)
02076         {
02077             double cx, cy, r;
02078 
02079             cx = ParseLengthOrPercentage(circleObject.Attributes?["cx"]?.Value, width);
02080             cy = ParseLengthOrPercentage(circleObject.Attributes?["cy"]?.Value, height);
02081             r = ParseLengthOrPercentage(circleObject.Attributes?["r"]?.Value, diagonal);
02082 
02083             string tag = circleObject.Attributes?["id"]?.Value;
02084 
02085             PresentationAttributes circleAttributes = InterpretPresentationAttributes(circleObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
02086 
02087             bool hadClippingPath = ApplyClipPath(circleObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
02088 
02089             if (circleAttributes.StrokeFirst)
02090             {
02091                 if (circleAttributes.Stroke != null)
02092                 {
02093                     Brush strokeColour =
      circleAttributes.Stroke.MultiplyOpacity(circleAttributes.Opacity * circleAttributes.StrokeOpacity);
02094 
02095                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02096                     GraphicsPath oldPath = path;
02097                     double[,] transform = null;
02098 
02099                     if (circleAttributes.StrokeGradientNeedsTransform)
02100                     {
02101                         (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(cx - r,
      cy - r, r * 2, r * 2));
02102                     }
02103 
02104                     if (transform != null)
02105                     {
02106                         gpr.Save();
02107                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02108                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02109                     }
02110 
02111                     gpr.StrokePath(path, strokeColour, circleAttributes.StrokeThickness,
      circleAttributes.LineCap, circleAttributes.LineJoin, circleAttributes.LineDash, tag:  tag);
02112 
02113                     if (transform != null)
02114                     {
02115                         gpr.Restore();
02116                         path = oldPath;
02117                     }
02118                 }
02119 
02120                 if (circleAttributes.Fill != null)
02121                 {
02122                     Brush fillColour = circleAttributes.Fill.MultiplyOpacity(circleAttributes.Opacity
      * circleAttributes.FillOpacity);
02123 
02124                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02125                     GraphicsPath oldPath = path;
02126                     double[,] transform = null;
02127 
02128                     if (circleAttributes.FillGradientNeedsTransform)
02129                     {
02130                         (fillColour, transform) = TransformBrush(fillColour, new Rectangle(cx - r, cy
      - r, r * 2, r * 2));
02131                     }
02132 
02133                     if (transform != null)
02134                     {
02135                         gpr.Save();
```

```
02136                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02137                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02138                     }
02139
02140                     gpr.FillPath(path, fillColour, tag:  tag);
02141
02142                     if (transform != null)
02143                     {
02144                         gpr.Restore();
02145                         path = oldPath;
02146                     }
02147                 }
02148             }
02149             else
02150             {
02151                 if (circleAttributes.Fill != null)
02152                 {
02153                     Brush fillColour = circleAttributes.Fill.MultiplyOpacity(circleAttributes.Opacity
      * circleAttributes.FillOpacity);
02154                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02155                     GraphicsPath oldPath = path;
02156                     double[,] transform = null;
02157
02158                     if (circleAttributes.FillGradientNeedsTransform)
02159                     {
02160                         (fillColour, transform) = TransformBrush(fillColour, new Rectangle(cx - r, cy
      - r, r * 2, r * 2));
02161                     }
02162
02163                     if (transform != null)
02164                     {
02165                         gpr.Save();
02166                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02167                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02168                     }
02169
02170                     gpr.FillPath(path, fillColour, tag:  tag);
02171
02172                     if (transform != null)
02173                     {
02174                         gpr.Restore();
02175                         path = oldPath;
02176                     }
02177                 }
02178
02179                 if (circleAttributes.Stroke != null)
02180                 {
02181                     Brush strokeColour =
      circleAttributes.Stroke.MultiplyOpacity(circleAttributes.Opacity * circleAttributes.StrokeOpacity);
02182                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02183                     GraphicsPath oldPath = path;
02184                     double[,] transform = null;
02185
02186                     if (circleAttributes.StrokeGradientNeedsTransform)
02187                     {
02188                         (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(cx - r,
      cy - r, r * 2, r * 2));
02189                     }
02190
02191                     if (transform != null)
02192                     {
02193                         gpr.Save();
02194                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02195                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02196                     }
02197
02198                     gpr.StrokePath(path, strokeColour, circleAttributes.StrokeThickness,
      circleAttributes.LineCap, circleAttributes.LineJoin, circleAttributes.LineDash, tag:  tag);
02199
02200                     if (transform != null)
02201                     {
02202                         gpr.Restore();
02203                         path = oldPath;
02204                     }
02205                 }
02206             }
02207
02208             if (hadClippingPath)
02209             {
02210                 gpr.Restore();
02211             }
02212
02213             if (circleAttributes.NeedsRestore)
02214             {
```

```
02215                 gpr.Restore();
02216             }
02217         }
02218 
02219         private static void InterpretEllipseObject(XmlNode currObject, Graphics gpr, double width,
      double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
      styleSheets, Dictionary<string, (bool, Brush)> gradients)
02220         {
02221             double cx, cy, rx, ry;
02222 
02223             cx = ParseLengthOrPercentage(currObject.Attributes?["cx"]?.Value, width);
02224             cy = ParseLengthOrPercentage(currObject.Attributes?["cy"]?.Value, height);
02225             rx = ParseLengthOrPercentage(currObject.Attributes?["rx"]?.Value, width, double.NaN);
02226             ry = ParseLengthOrPercentage(currObject.Attributes?["ry"]?.Value, height, double.NaN);
02227 
02228             string tag = currObject.Attributes?["id"]?.Value;
02229 
02230             if (double.IsNaN(rx) && !double.IsNaN(ry))
02231             {
02232                 rx = ry;
02233             }
02234             else if (!double.IsNaN(rx) && double.IsNaN(ry))
02235             {
02236                 ry = rx;
02237             }
02238 
02239             if (rx > 0 && ry > 0)
02240             {
02241 
02242                 PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
02243 
02244                 bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
02245 
02246                 double r = Math.Min(rx, ry);
02247 
02248                 gpr.Save();
02249                 gpr.Translate(cx, cy);
02250                 gpr.Scale(rx / r, ry / r);
02251 
02252                 if (currAttributes.StrokeFirst)
02253                 {
02254                     if (currAttributes.Stroke != null)
02255                     {
02256                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
02257 
02258                         GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02259                         GraphicsPath oldPath = path;
02260                         double[,] transform = null;
02261 
02262                         if (currAttributes.StrokeGradientNeedsTransform)
02263                         {
02264                             (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(0,
      0, r * 2, r * 2));
02265                         }
02266 
02267                         if (transform != null)
02268                         {
02269                             gpr.Save();
02270                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02271                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02272                         }
02273 
02274                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02275 
02276                         if (transform != null)
02277                         {
02278                             gpr.Restore();
02279                             path = oldPath;
02280                         }
02281                     }
02282 
02283                     if (currAttributes.Fill != null)
02284                     {
02285                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
02286 
02287                         GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02288                         GraphicsPath oldPath = path;
02289                         double[,] transform = null;
02290 
02291                         if (currAttributes.FillGradientNeedsTransform)
02292                         {
```

```
02293                           (fillColour, transform) = TransformBrush(fillColour, new Rectangle(0, 0, r
      * 2, r * 2));
02294                       }
02295
02296                       if (transform != null)
02297                       {
02298                           gpr.Save();
02299                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02300                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02301                       }
02302
02303                       gpr.StrokePath(path, fillColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02304
02305                       if (transform != null)
02306                       {
02307                           gpr.Restore();
02308                           path = oldPath;
02309                       }
02310                   }
02311               }
02312               else
02313               {
02314                   if (currAttributes.Fill != null)
02315                   {
02316                       Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
02317
02318                       GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02319                       GraphicsPath oldPath = path;
02320                       double[,] transform = null;
02321
02322                       if (currAttributes.FillGradientNeedsTransform)
02323                       {
02324                           (fillColour, transform) = TransformBrush(fillColour, new Rectangle(0, 0, r
      * 2, r * 2));
02325                       }
02326
02327                       if (transform != null)
02328                       {
02329                           gpr.Save();
02330                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02331                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02332                       }
02333
02334                       gpr.StrokePath(path, fillColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02335
02336                       if (transform != null)
02337                       {
02338                           gpr.Restore();
02339                           path = oldPath;
02340                       }
02341                   }
02342
02343                   if (currAttributes.Stroke != null)
02344                   {
02345                       Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
02346                       GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02347                       GraphicsPath oldPath = path;
02348                       double[,] transform = null;
02349
02350                       if (currAttributes.StrokeGradientNeedsTransform)
02351                       {
02352                           (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(0,
      0, r * 2, r * 2));
02353                       }
02354
02355                       if (transform != null)
02356                       {
02357                           gpr.Save();
02358                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02359                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02360                       }
02361
02362                       gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02363
02364                       if (transform != null)
02365                       {
02366                           gpr.Restore();
02367                           path = oldPath;
02368                       }
```

```
02369                   }
02370               }
02371
02372               gpr.Restore();
02373
02374               if (hadClippingPath)
02375               {
02376                   gpr.Restore();
02377               }
02378
02379               if (currAttributes.NeedsRestore)
02380               {
02381                   gpr.Restore();
02382               }
02383           }
02384       }
02385
02386       private static void InterpretLineObject(XmlNode lineObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients)
02387       {
02388           double x1, x2, y1, y2;
02389
02390           x1 = ParseLengthOrPercentage(lineObject.Attributes?["x1"]?.Value, width);
02391           y1 = ParseLengthOrPercentage(lineObject.Attributes?["y1"]?.Value, height);
02392           x2 = ParseLengthOrPercentage(lineObject.Attributes?["x2"]?.Value, width);
02393           y2 = ParseLengthOrPercentage(lineObject.Attributes?["y2"]?.Value, height);
02394
02395           PresentationAttributes lineAttributes = InterpretPresentationAttributes(lineObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
02396
02397           bool hadClippingPath = ApplyClipPath(lineObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
02398
02399           string tag = lineObject.Attributes?["id"]?.Value;
02400
02401           if (lineAttributes.Stroke != null)
02402           {
02403               Brush strokeColour = lineAttributes.Stroke.MultiplyOpacity(lineAttributes.Opacity *
      lineAttributes.StrokeOpacity);
02404
02405               GraphicsPath path = new GraphicsPath().MoveTo(x1, y1).LineTo(x2, y2);
02406               GraphicsPath oldPath = path;
02407               double[,] transform = null;
02408
02409               if (lineAttributes.StrokeGradientNeedsTransform)
02410               {
02411                   (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(x1, y1, x2
      - x1, y2 - y1));
02412               }
02413
02414               if (transform != null)
02415               {
02416                   gpr.Save();
02417                   gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1, 1],
      transform[0, 2], transform[1, 2]);
02418                   path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02419               }
02420
02421               gpr.StrokePath(path, strokeColour, lineAttributes.StrokeThickness,
      lineAttributes.LineCap, lineAttributes.LineJoin, lineAttributes.LineDash, tag:  tag);
02422
02423               if (transform != null)
02424               {
02425                   gpr.Restore();
02426                   path = oldPath;
02427               }
02428           }
02429
02430           if (hadClippingPath)
02431           {
02432               gpr.Restore();
02433           }
02434
02435           if (lineAttributes.NeedsRestore)
02436           {
02437               gpr.Restore();
02438           }
02439       }
02440
02441       private static void SetStyleAttributes(XmlNode obj, IEnumerable<Stylesheet> styleSheets)
02442       {
02443           string style = obj.Attributes?["style"]?.Value;
02444
02445           string classes = obj.Attributes?["class"]?.Value;
02446
02447           if (!string.IsNullOrEmpty(classes))
```

```
02448             {
02449                 string[] splitClasses = classes.Split(' ');
02450 
02451                 foreach (string className in splitClasses)
02452                 {
02453                     if (!string.IsNullOrEmpty(className.Trim()))
02454                     {
02455                         foreach (Stylesheet sheet in styleSheets)
02456                         {
02457                             foreach (StyleRule rule in sheet.StyleRules)
02458                             {
02459                                 if (rule.SelectorText.Contains("." + className))
02460                                 {
02461                                     style = rule.Style.CssText + "; " + style;
02462                                 }
02463                             }
02464                         }
02465                     }
02466                 }
02467             }
02468 
02469             if (!string.IsNullOrEmpty(style))
02470             {
02471                 string[] splitStyle = style.Split(';');
02472 
02473                 for (int i = 0; i < splitStyle.Length; i++)
02474                 {
02475                     string[] styleCouple = splitStyle[i].Split(':');
02476 
02477                     if (styleCouple.Length == 2)
02478                     {
02479                         string styleName = styleCouple[0].Trim();
02480                         string styleValue = styleCouple[1].Trim();
02481 
02482                         ((XmlElement)obj).SetAttribute(styleName, styleValue);
02483                     }
02484                     else if (!string.IsNullOrWhiteSpace(splitStyle[i]))
02485                     {
02486                         throw new InvalidOperationException("The style specification is not valid:  "
      + splitStyle[i]);
02487                     }
02488                 }
02489             }
02490         }
02491 
02492         internal static PresentationAttributes InterpretPresentationAttributes(XmlNode obj,
      PresentationAttributes parentPresentationAttributes, double width, double height, double diagonal,
      Graphics gpr, IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients)
02493         {
02494             SetStyleAttributes(obj, styleSheets);
02495 
02496             PresentationAttributes tbr = parentPresentationAttributes.Clone();
02497 
02498             string stroke = obj.Attributes?["stroke"]?.Value;
02499             string strokeOpacity = obj.Attributes?["stroke-opacity"]?.Value;
02500             string fill = obj.Attributes?["fill"]?.Value;
02501             string fillOpacity = obj.Attributes?["fill-opacity"]?.Value;
02502             string currentColour = obj.Attributes?["colour"]?.Value;
02503             string strokeThickness = obj.Attributes?["stroke-width"]?.Value;
02504             string lineCap = obj.Attributes?["stroke-linecap"]?.Value;
02505             string lineJoin = obj.Attributes?["stroke-linejoin"]?.Value;
02506             string opacity = obj.Attributes?["opacity"]?.Value;
02507             string strokeDashArray = obj.Attributes?["stroke-dasharray"]?.Value;
02508             string strokeDashOffset = obj.Attributes?["stroke-dashoffset"]?.Value;
02509             string paintOrder = obj.Attributes?["paint-order"]?.Value;
02510 
02511 
02512             string xA = obj.Attributes?["x"]?.Value;
02513             string yA = obj.Attributes?["y"]?.Value;
02514             string wA = obj.Attributes?["width"]?.Value;
02515             string hA = obj.Attributes?["height"]?.Value;
02516 
02517             string transform = obj.Attributes?["transform"]?.Value;
02518 
02519             if (xA != null)
02520             {
02521                 tbr.X = ParseLengthOrPercentage(xA, width);
02522             }
02523 
02524             if (yA != null)
02525             {
02526                 tbr.Y = ParseLengthOrPercentage(yA, height);
02527             }
02528 
02529             if (wA != null)
02530             {
02531                 tbr.Width = ParseLengthOrPercentage(wA, width);
```

```
02532             }
02533
02534             if (hA != null)
02535             {
02536                 tbr.Height = ParseLengthOrPercentage(hA, height);
02537             }
02538
02539             if (stroke != null)
02540             {
02541                 if (stroke.Trim().StartsWith("url("))
02542                 {
02543                     string url = stroke.Trim().Substring(4);
02544                     if (url.EndsWith(")"))
02545                     {
02546                         url = url.Substring(0, url.Length - 1);
02547                     }
02548
02549                     url = url.Trim();
02550
02551                     if (url.StartsWith("#"))
02552                     {
02553                         url = url.Substring(1);
02554                         if (gradients.TryGetValue(url, out (bool, Brush) brush))
02555                         {
02556                             tbr.Stroke = brush.Item2;
02557                             tbr.StrokeGradientNeedsTransform = !brush.Item1;
02558                         }
02559                     }
02560                     else
02561                     {
02562                         tbr.Stroke = null;
02563                     }
02564                 }
02565                 else
02566                 {
02567                     tbr.Stroke = Colour.FromCSSString(stroke);
02568                 }
02569             }
02570
02571             if (strokeOpacity != null)
02572             {
02573                 tbr.StrokeOpacity = ParseLengthOrPercentage(strokeOpacity, 1);
02574             }
02575
02576             if (fill != null)
02577             {
02578                 if (fill.Trim().StartsWith("url("))
02579                 {
02580                     string url = fill.Trim().Substring(4);
02581                     if (url.EndsWith(")"))
02582                     {
02583                         url = url.Substring(0, url.Length - 1);
02584                     }
02585
02586                     url = url.Trim();
02587
02588                     if (url.StartsWith("#"))
02589                     {
02590                         url = url.Substring(1);
02591                         if (gradients.TryGetValue(url, out (bool, Brush) brush))
02592                         {
02593                             tbr.Fill = brush.Item2;
02594                             tbr.FillGradientNeedsTransform = !brush.Item1;
02595                         }
02596                     }
02597                     else
02598                     {
02599                         tbr.Fill = null;
02600                     }
02601                 }
02602                 else
02603                 {
02604                     tbr.Fill = Colour.FromCSSString(fill);
02605                 }
02606             }
02607
02608             if (fillOpacity != null)
02609             {
02610                 tbr.FillOpacity = ParseLengthOrPercentage(fillOpacity, 1);
02611             }
02612
02613             if (currentColour != null)
02614             {
02615                 tbr.CurrentColour = Colour.FromCSSString(currentColour);
02616             }
02617
02618             if (strokeThickness != null)
```

```
02619             {
02620                 tbr.StrokeThickness = ParseLengthOrPercentage(strokeThickness, diagonal);
02621             }
02622 
02623             if (lineCap != null)
02624             {
02625                 if (lineCap.Equals("butt", StringComparison.OrdinalIgnoreCase))
02626                 {
02627                     tbr.LineCap = LineCaps.Butt;
02628                 }
02629                 else if (lineCap.Equals("round", StringComparison.OrdinalIgnoreCase))
02630                 {
02631                     tbr.LineCap = LineCaps.Round;
02632                 }
02633                 else if (lineCap.Equals("square", StringComparison.OrdinalIgnoreCase))
02634                 {
02635                     tbr.LineCap = LineCaps.Square;
02636                 }
02637             }
02638 
02639             if (lineJoin != null)
02640             {
02641                 if (lineJoin.Equals("bevel", StringComparison.OrdinalIgnoreCase))
02642                 {
02643                     tbr.LineJoin = LineJoins.Bevel;
02644                 }
02645                 else if (lineJoin.Equals("miter", StringComparison.OrdinalIgnoreCase) ||
      lineJoin.Equals("miter-clip", StringComparison.OrdinalIgnoreCase))
02646                 {
02647                     tbr.LineJoin = LineJoins.Miter;
02648                 }
02649                 else if (lineJoin.Equals("round", StringComparison.OrdinalIgnoreCase))
02650                 {
02651                     tbr.LineJoin = LineJoins.Round;
02652                 }
02653             }
02654 
02655             if (opacity != null)
02656             {
02657                 tbr.Opacity = ParseLengthOrPercentage(opacity, 1);
02658             }
02659 
02660             if (strokeDashArray != null)
02661             {
02662                 if (strokeDashArray != "none")
02663                 {
02664                     double[] parsedArray = ParseListOfDoubles(strokeDashArray);
02665 
02666                     tbr.LineDash = new LineDash(parsedArray[0], parsedArray.Length > 1 ?
      parsedArray[1] :  parsedArray[0], tbr.LineDash.Phase);
02667                 }
02668                 else
02669                 {
02670                     tbr.LineDash = LineDash.SolidLine;
02671                 }
02672             }
02673 
02674             if (strokeDashOffset != null)
02675             {
02676                 tbr.LineDash = new LineDash(tbr.LineDash.UnitsOn, tbr.LineDash.UnitsOff,
      ParseLengthOrPercentage(strokeDashOffset, diagonal));
02677             }
02678 
02679             if (paintOrder != null)
02680             {
02681                 if (paintOrder.Equals("normal", StringComparison.OrdinalIgnoreCase))
02682                 {
02683                     tbr.StrokeFirst = false;
02684                 }
02685                 else
02686                 {
02687                     if (paintOrder.IndexOf("stroke", StringComparison.OrdinalIgnoreCase) >= 0 &&
      (paintOrder.IndexOf("fill", StringComparison.OrdinalIgnoreCase) < 0 || paintOrder.IndexOf("fill",
      StringComparison.OrdinalIgnoreCase) > paintOrder.IndexOf("stroke",
      StringComparison.OrdinalIgnoreCase)))
02688                     {
02689                         tbr.StrokeFirst = true;
02690                     }
02691                     else
02692                     {
02693                         tbr.StrokeFirst = false;
02694                     }
02695                 }
02696             }
02697 
02698             if (transform != null)
02699             {
```

```
02700                 gpr.Save();
02701                 tbr.NeedsRestore = true;
02702 
02703                 string[] transforms = ParseListOfTransforms(transform);
02704 
02705                 for (int i = 0; i < transforms.Length; i++)
02706                 {
02707                     if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
02708                     {
02709                         double a = ParseLengthOrPercentage(transforms[i + 1], 1);
02710                         double b = ParseLengthOrPercentage(transforms[i + 2], 1);
02711                         double c = ParseLengthOrPercentage(transforms[i + 3], 1);
02712                         double d = ParseLengthOrPercentage(transforms[i + 4], 1);
02713                         double e = ParseLengthOrPercentage(transforms[i + 5], 1);
02714                         double f = ParseLengthOrPercentage(transforms[i + 6], 1);
02715 
02716                         gpr.Transform(a, b, c, d, e, f);
02717                         i += 6;
02718                     }
02719                     else if (transforms[i].Equals("translate", StringComparison.OrdinalIgnoreCase))
02720                     {
02721                         double x = ParseLengthOrPercentage(transforms[i + 1], 1);
02722 
02723                         double y;
02724 
02725                         if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
02726                         {
02727                             gpr.Translate(x, y);
02728                             i += 2;
02729                         }
02730                         else
02731                         {
02732                             gpr.Translate(x, 0);
02733                             i++;
02734                         }
02735                     }
02736                     else if (transforms[i].Equals("scale", StringComparison.OrdinalIgnoreCase))
02737                     {
02738                         double x = ParseLengthOrPercentage(transforms[i + 1], 1);
02739 
02740                         double y;
02741 
02742                         if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
02743                         {
02744                             gpr.Scale(x, y);
02745                             i += 2;
02746                         }
02747                         else
02748                         {
02749                             gpr.Scale(x, x);
02750                             i++;
02751                         }
02752                     }
02753                     else if (transforms[i].Equals("rotate", StringComparison.OrdinalIgnoreCase))
02754                     {
02755                         double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02756 
02757                         double x, y;
02758 
02759                         if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
02760                         {
02761                             gpr.RotateAt(a, new Point(x, y));
02762                             i += 2;
02763                         }
02764                         else
02765                         {
02766                             gpr.Rotate(a);
02767                             i++;
02768                         }
02769                     }
02770                     else if (transforms[i].Equals("skewX", StringComparison.OrdinalIgnoreCase))
02771                     {
02772                         double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02773 
02774                         gpr.Transform(1, 0, Math.Tan(psi), 1, 0, 0);
02775 
02776                         i++;
02777                     }
02778                     else if (transforms[i].Equals("skewY", StringComparison.OrdinalIgnoreCase))
02779                     {
02780                         double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02781 
02782                         gpr.Transform(1, Math.Tan(psi), 0, 1, 0, 0);
```

```
02783
02784                       i++;
02785                   }
02786               }
02787           }
02788
02789           return tbr;
02790       }
02791
02792       private static double ParseLengthOrPercentage(string value, double total, double defaultValue
      = 0)
02793       {
02794           if (value != null)
02795           {
02796               if (value.Contains("%"))
02797               {
02798                   value = value.Replace("%", "");
02799                   return double.Parse(value, System.Globalization.CultureInfo.InvariantCulture) *
      total / 100;
02800               }
02801               else if (double.TryParse(value.Replace("px", "").Replace("pt", ""),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double
      result))
02802               {
02803                   return result;
02804               }
02805               else
02806               {
02807                   string cleanedNumber = Regexes.NumberRegex.Match(value).Value;
02808                   return double.Parse(cleanedNumber,
      System.Globalization.CultureInfo.InvariantCulture);
02809               }
02810           }
02811           else
02812           {
02813               return defaultValue;
02814           }
02815       }
02816
02817       internal class PresentationAttributes
02818       {
02819           public Dictionary<string, FontFamily> EmbeddedFonts;
02820
02821           public Brush Stroke = null;
02822           public double StrokeOpacity = 1;
02823           public Brush Fill = Colour.FromRgb(0, 0, 0);
02824           public double FillOpacity = 1;
02825           public Brush CurrentColour = null;
02826           public double StrokeThickness = 1;
02827           public LineCaps LineCap = LineCaps.Butt;
02828           public LineJoins LineJoin = LineJoins.Miter;
02829           public double Opacity = 1;
02830           public LineDash LineDash = new LineDash(0, 0, 0);
02831           public bool NeedsRestore = false;
02832           public bool StrokeFirst = false;
02833           public double X;
02834           public double Y;
02835           public double Width;
02836           public double Height;
02837           public bool StrokeGradientNeedsTransform = false;
02838           public bool FillGradientNeedsTransform = false;
02839
02840           public PresentationAttributes Clone()
02841           {
02842               return new PresentationAttributes()
02843               {
02844                   EmbeddedFonts = this.EmbeddedFonts,
02845
02846                   Stroke = this.Stroke,
02847                   StrokeOpacity = this.StrokeOpacity,
02848                   Fill = this.Fill,
02849                   FillOpacity = this.FillOpacity,
02850                   CurrentColour = this.CurrentColour,
02851                   StrokeThickness = this.StrokeThickness,
02852                   LineCap = this.LineCap,
02853                   LineJoin = this.LineJoin,
02854                   Opacity = this.Opacity,
02855                   LineDash = this.LineDash,
02856                   StrokeFirst = this.StrokeFirst,
02857                   X = this.X,
02858                   Y = this.Y,
02859                   Width = this.Width,
02860                   Height = this.Height
02861               };
02862           }
02863       }
02864
```

```
02865         private static class Regexes
02866         {
02867             public static Regex ListSeparator = new Regex("[ \\t\\n\\r\\f]*,[ \\t\\n\\r\\f]*|[
      \\t\\n\\r\\f]+", RegexOptions.Compiled);
02868             public static Regex FontFamilySeparator = new Regex("(?:^|,)(\"(?:[^\"])*\"|[^,]*)",
      RegexOptions.Compiled);
02869             public static Regex NumberRegex = new Regex(@"^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?",
      RegexOptions.Compiled);
02870         }
02871 
02872         private static double[] ParseListOfDoubles(string value)
02873         {
02874             if (value == null)
02875             {
02876                 return null;
02877             }
02878 
02879             string[] splitValue = Regexes.ListSeparator.Split(value);
02880             double[] tbr = new double[splitValue.Length];
02881 
02882             for (int i = 0; i < splitValue.Length; i++)
02883             {
02884                 tbr[i] = double.Parse(splitValue[i],
      System.Globalization.CultureInfo.InvariantCulture);
02885             }
02886 
02887             return tbr;
02888         }
02889 
02890         private static string[] ParseListOfTransforms(string value)
02891         {
02892             if (value == null)
02893             {
02894                 return null;
02895             }
02896 
02897             string[] splitValue = Regexes.ListSeparator.Split(value.Replace("(", " ").Replace(")", "
      ").Trim());
02898 
02899             return splitValue;
02900         }
02901 
02902         private static List<KeyValuePair<string, FontFamily» GetEmbeddedFonts(string styleBlock)
02903         {
02904             StringReader sr = new StringReader(styleBlock);
02905 
02906             List<KeyValuePair<string, FontFamily» tbr = new List<KeyValuePair<string, FontFamily»();
02907 
02908             while (sr.Peek() >= 0)
02909             {
02910                 string token = ReadCSSToken(sr);
02911 
02912                 if (token.Equals("@font-face", StringComparison.OrdinalIgnoreCase))
02913                 {
02914                     List<string> tokens = new List<string>();
02915 
02916                     while (!token.Equals("}", StringComparison.OrdinalIgnoreCase))
02917                     {
02918                         token = ReadCSSToken(sr);
02919                         tokens.Add(token);
02920                     }
02921 
02922                     KeyValuePair<string, FontFamily>?  fontFace = ParseFontFaceBlock(tokens);
02923 
02924                     if (fontFace != null)
02925                     {
02926                         tbr.Add(fontFace.Value);
02927                     }
02928                 }
02929             }
02930 
02931             return tbr;
02932         }
02933 
02934         private static IEnumerable<KeyValuePair<string, IFilter» GetFilters(XmlNode definitionsNode,
      List<Stylesheet> styleSheets)
02935         {
02936             Dictionary<string, IFilter> tbr = new Dictionary<string, IFilter>();
02937 
02938             foreach (XmlNode definition in definitionsNode.ChildNodes)
02939             {
02940                 if (definition.Name.Equals("filter", StringComparison.OrdinalIgnoreCase))
02941                 {
02942                     XmlElement filter = (XmlElement)definition;
02943 
02944                     string id = filter.GetAttribute("id");
02945 
```

```
02946                   List<ILocationInvariantFilter> filterElements = new
      List<ILocationInvariantFilter>();
02947
02948                   foreach (XmlNode filterDefinition in definition.ChildNodes)
02949                   {
02950                       if (filterDefinition.Name.Equals("feGaussianBlur",
      StringComparison.OrdinalIgnoreCase))
02951                       {
02952                           XmlElement actualFilter = (XmlElement)filterDefinition;
02953
02954                           string stdDeviation = actualFilter.GetAttribute("stdDeviation");
02955
02956                           filterElements.Add(new GaussianBlurFilter(double.Parse(stdDeviation,
      System.Globalization.CultureInfo.InvariantCulture)));
02957                       }
02958                       else if (filterDefinition.Name.Equals("feColorMatrix",
      StringComparison.OrdinalIgnoreCase))
02959                       {
02960                           XmlElement actualFilter = (XmlElement)filterDefinition;
02961
02962                           string type = actualFilter.GetAttribute("type");
02963
02964                           if (type.Equals("matrix", StringComparison.OrdinalIgnoreCase))
02965                           {
02966                               string values = actualFilter.GetAttribute("values");
02967                               double[] parsedValues = (from el in
      System.Text.RegularExpressions.Regex.Split(values, "\\s") select double.Parse(el.Trim(),
      System.Globalization.CultureInfo.InvariantCulture)).ToArray();
02968
02969                               if (parsedValues.Length == 20)
02970                               {
02971                                   double[,] matrix = new double[5, 5];
02972                                   matrix[4, 4] = 1;
02973
02974                                   for (int i = 0; i < 20; i++)
02975                                   {
02976                                       int y = i / 5;
02977                                       int x = i % 5;
02978
02979                                       matrix[y, x] = parsedValues[i];
02980                                   }
02981
02982                                   filterElements.Add(new ColourMatrixFilter(new
      ColourMatrix(matrix)));
02983                               }
02984                           }
02985                       }
02986                   }
02987
02988                   if (filterElements.Count > 0)
02989                   {
02990                       if (filterElements.Count == 1)
02991                       {
02992                           tbr.Add(id, filterElements[0]);
02993                       }
02994                       else
02995                       {
02996                           tbr.Add(id, new CompositeLocationInvariantFilter(filterElements));
02997                       }
02998                   }
02999               }
03000           }
03001
03002           return tbr;
03003       }
03004
03005       private static IEnumerable<KeyValuePair<string, XmlNode» GetMasks(XmlNode definitionsNode,
      List<Stylesheet> styleSheets)
03006       {
03007           Dictionary<string, XmlNode> tbr = new Dictionary<string, XmlNode>();
03008
03009           foreach (XmlNode definition in definitionsNode.ChildNodes)
03010           {
03011               if (definition.Name.Equals("mask", StringComparison.OrdinalIgnoreCase))
03012               {
03013                   XmlElement mask = (XmlElement)definition;
03014
03015                   string id = mask.GetAttribute("id");
03016
03017                   tbr.Add(id, definition);
03018               }
03019           }
03020
03021           return tbr;
03022       }
03023
03024       private static IEnumerable<KeyValuePair<string, (bool, Brush)» GetGradients(XmlNode
```

```
    definitionsNode, List<Stylesheet> styleSheets)
03025         {
03026             Dictionary<string, (bool, Brush)> tbr = new Dictionary<string, (bool, Brush)>();
03027             Dictionary<string, XmlNodeList> stopLists = new Dictionary<string, XmlNodeList>();
03028 
03029             foreach (XmlNode definition in definitionsNode.ChildNodes)
03030             {
03031                 if (definition.Name.Equals("linearGradient", StringComparison.OrdinalIgnoreCase))
03032                 {
03033                     XmlElement gradient = (XmlElement)definition;
03034 
03035                     string id = gradient.GetAttribute("id");
03036 
03037                     bool userSpaceOnUse = gradient.GetAttribute("gradientUnits") == "userSpaceOnUse";
03038 
03039                     double x1;
03040                     double y1;
03041                     double x2;
03042                     double y2;
03043 
03044                     if (!(gradient.HasAttribute("x1") && double.TryParse(gradient.GetAttribute("x1"),
    System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out x1)))
03045                     {
03046                         x1 = 0;
03047                     }
03048 
03049                     if (!(gradient.HasAttribute("y1") && double.TryParse(gradient.GetAttribute("y1"),
    System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out y1)))
03050                     {
03051                         y1 = 0;
03052                     }
03053 
03054                     if (!(gradient.HasAttribute("x2") && double.TryParse(gradient.GetAttribute("x2"),
    System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out x2)))
03055                     {
03056                         x2 = 0;
03057                     }
03058 
03059                     if (!(gradient.HasAttribute("y2") && double.TryParse(gradient.GetAttribute("y2"),
    System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out y2)))
03060                     {
03061                         y2 = 0;
03062                     }
03063 
03064                     List<GradientStop> gradientStops = new List<GradientStop>();
03065 
03066                     XmlNodeList childNodes;
03067 
03068                     if (gradient.HasAttribute("xlink:href"))
03069                     {
03070                         string refId = gradient.GetAttribute("xlink:href").Trim();
03071 
03072                         if (refId.StartsWith("#"))
03073                         {
03074                             refId = refId.Substring(1);
03075 
03076                             if (!stopLists.TryGetValue(refId, out childNodes))
03077                             {
03078                                 childNodes = gradient.ChildNodes;
03079                             }
03080                         }
03081                         else
03082                         {
03083                             childNodes = gradient.ChildNodes;
03084                         }
03085                     }
03086                     else
03087                     {
03088                         childNodes = gradient.ChildNodes;
03089                     }
03090 
03091                     stopLists[id] = childNodes;
03092 
03093                     foreach (XmlNode stopNode in childNodes)
03094                     {
03095                         if (stopNode.Name.Equals("stop", StringComparison.OrdinalIgnoreCase))
03096                         {
03097                             SetStyleAttributes(stopNode, styleSheets);
03098 
03099                             XmlElement stop = (XmlElement)stopNode;
03100 
03101                             double offset = 0;
03102                             double opacity = 1;
03103 
03104                             if (stop.HasAttribute("offset"))
03105                             {
03106                                 offset = ParseLengthOrPercentage(stop.GetAttribute("offset"), 1);
```

```
03107                               }
03108
03109                               if (stop.HasAttribute("stop-opacity"))
03110                               {
03111                                   opacity = ParseLengthOrPercentage(stop.GetAttribute("stop-opacity"),
      1);
03112                               }
03113
03114                               Colour stopColour = Colour.FromRgba(0, 0, 0, 0);
03115
03116                               if (stop.HasAttribute("stop-color"))
03117                               {
03118                                   stopColour = (Colour.FromCSSString(stop.GetAttribute("stop-color")) ??
      stopColour).WithAlpha(opacity);
03119                               }
03120
03121                               gradientStops.Add(new GradientStop(stopColour, offset));
03122                           }
03123                       }
03124
03125                       if (gradient.HasAttribute("gradientTransform"))
03126                       {
03127                           string transform = gradient.GetAttribute("gradientTransform");
03128                           string[] transforms = ParseListOfTransforms(transform);
03129
03130                           double[,] transformMatrix = MatrixUtils.Identity;
03131
03132                           for (int i = 0; i < transforms.Length; i++)
03133                           {
03134                               if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
03135                               {
03136                                   double a = ParseLengthOrPercentage(transforms[i + 1], 1);
03137                                   double b = ParseLengthOrPercentage(transforms[i + 2], 1);
03138                                   double c = ParseLengthOrPercentage(transforms[i + 3], 1);
03139                                   double d = ParseLengthOrPercentage(transforms[i + 4], 1);
03140                                   double e = ParseLengthOrPercentage(transforms[i + 5], 1);
03141                                   double f = ParseLengthOrPercentage(transforms[i + 6], 1);
03142
03143                                   transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { a, c, e }, { b, d, f }, { 0, 0, 1 } });
03144
03145                                   i += 6;
03146                               }
03147                               else if (transforms[i].Equals("translate",
      StringComparison.OrdinalIgnoreCase))
03148                               {
03149                                   double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03150
03151                                   double y;
03152
03153                                   if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03154                                   {
03155                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03156                                       i += 2;
03157                                   }
03158                                   else
03159                                   {
03160                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, 0);
03161                                       i++;
03162                                   }
03163                               }
03164                               else if (transforms[i].Equals("scale",
      StringComparison.OrdinalIgnoreCase))
03165                               {
03166                                   double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03167
03168                                   double y;
03169
03170                                   if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03171                                   {
03172                                       transformMatrix = MatrixUtils.Scale(transformMatrix, x, y);
03173                                       i += 2;
03174                                   }
03175                                   else
03176                                   {
03177                                       transformMatrix = MatrixUtils.Scale(transformMatrix, x, x);
03178                                       i++;
03179                                   }
03180                               }
03181                               else if (transforms[i].Equals("rotate",
      StringComparison.OrdinalIgnoreCase))
03182                               {
03183                                   double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03184
```

```
03185                                 double x, y;
03186
03187                                 if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
03188                                 {
03189                                     transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03190                                     transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03191                                     transformMatrix = MatrixUtils.Translate(transformMatrix, -x, -y);
03192                                     i += 2;
03193                                 }
03194                                 else
03195                                 {
03196                                     transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03197                                     i++;
03198                                 }
03199                             }
03200                             else if (transforms[i].Equals("skewX",
      StringComparison.OrdinalIgnoreCase))
03201                             {
03202                                 double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03203
03204                                 transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, Math.Tan(psi), 0 }, { 0, 1, 0 }, { 0, 0, 1 } });
03205                                 i++;
03206                             }
03207                             else if (transforms[i].Equals("skewY",
      StringComparison.OrdinalIgnoreCase))
03208                             {
03209                                 double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03210
03211                                 transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, 0, 0 }, { Math.Tan(psi), 1, 0 }, { 0, 0, 1 } });
03212                                 i++;
03213                             }
03214                         }
03215
03216                         double[] start = MatrixUtils.Multiply(transformMatrix, new double[] { x1, y1
      });
03217                         double[] end = MatrixUtils.Multiply(transformMatrix, new double[] { x2, y2 });
03218
03219                         x1 = start[0];
03220                         y1 = start[1];
03221                         x2 = end[0];
03222                         y2 = end[1];
03223                     }
03224
03225                     tbr.Add(id, (userSpaceOnUse, new LinearGradientBrush(new Point(x1, y1), new
      Point(x2, y2), gradientStops)));
03226                 }
03227                 else if (definition.Name.Equals("radialGradient", StringComparison.OrdinalIgnoreCase))
03228                 {
03229                     XmlElement gradient = (XmlElement)definition;
03230
03231                     string id = gradient.GetAttribute("id");
03232
03233                     bool userSpaceOnUse = gradient.GetAttribute("gradientUnits") == "userSpaceOnUse";
03234
03235                     double cx;
03236                     double cy;
03237                     double r;
03238
03239                     if (!(gradient.HasAttribute("cx") && double.TryParse(gradient.GetAttribute("cx"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out cx)))
03240                     {
03241                         cx = 0;
03242                     }
03243
03244                     if (!(gradient.HasAttribute("cy") && double.TryParse(gradient.GetAttribute("cy"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out cy)))
03245                     {
03246                         cy = 0;
03247                     }
03248
03249                     if (!(gradient.HasAttribute("r") && double.TryParse(gradient.GetAttribute("r"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out r)))
03250                     {
03251                         r = 0;
03252                     }
03253
03254                     double fx;
03255                     double fy;
03256
03257                     if (!(gradient.HasAttribute("fx") && double.TryParse(gradient.GetAttribute("fx"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out fx)))
```

```
03258                     {
03259                         fx = cx;
03260                     }
03261 
03262                     if (!(gradient.HasAttribute("fy") && double.TryParse(gradient.GetAttribute("fy"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out fy)))
03263                     {
03264                         fy = cy;
03265                     }
03266 
03267                     List<GradientStop> gradientStops = new List<GradientStop>();
03268 
03269                     XmlNodeList childNodes;
03270 
03271                     if (gradient.HasAttribute("xlink:href"))
03272                     {
03273                         string refId = gradient.GetAttribute("xlink:href").Trim();
03274 
03275                         if (refId.StartsWith("#"))
03276                         {
03277                             refId = refId.Substring(1);
03278 
03279                             if (!stopLists.TryGetValue(refId, out childNodes))
03280                             {
03281                                 childNodes = gradient.ChildNodes;
03282                             }
03283                         }
03284                         else
03285                         {
03286                             childNodes = gradient.ChildNodes;
03287                         }
03288                     }
03289                     else
03290                     {
03291                         childNodes = gradient.ChildNodes;
03292                     }
03293 
03294                     stopLists[id] = childNodes;
03295 
03296                     foreach (XmlNode stopNode in childNodes)
03297                     {
03298                         if (stopNode.Name.Equals("stop", StringComparison.OrdinalIgnoreCase))
03299                         {
03300                             SetStyleAttributes(stopNode, styleSheets);
03301 
03302                             XmlElement stop = (XmlElement)stopNode;
03303 
03304                             double offset = 0;
03305                             double opacity = 1;
03306 
03307                             if (stop.HasAttribute("offset"))
03308                             {
03309                                 offset = ParseLengthOrPercentage(stop.GetAttribute("offset"), 1);
03310                             }
03311 
03312                             if (stop.HasAttribute("stop-opacity"))
03313                             {
03314                                 opacity = ParseLengthOrPercentage(stop.GetAttribute("stop-opacity"),
      1);
03315                             }
03316 
03317                             Colour stopColour = Colour.FromRgba(0, 0, 0, 0);
03318 
03319                             if (stop.HasAttribute("stop-color"))
03320                             {
03321                                 stopColour = (Colour.FromCSSString(stop.GetAttribute("stop-color")) ??
      stopColour).WithAlpha(opacity);
03322                             }
03323 
03324                             gradientStops.Add(new GradientStop(stopColour, offset));
03325                         }
03326                     }
03327 
03328                     if (gradient.HasAttribute("gradientTransform"))
03329                     {
03330                         string transform = gradient.GetAttribute("gradientTransform");
03331                         string[] transforms = ParseListOfTransforms(transform);
03332 
03333                         double[,] transformMatrix = MatrixUtils.Identity;
03334 
03335                         for (int i = 0; i < transforms.Length; i++)
03336                         {
03337                             if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
03338                             {
03339                                 double a = ParseLengthOrPercentage(transforms[i + 1], 1);
03340                                 double b = ParseLengthOrPercentage(transforms[i + 2], 1);
03341                                 double c = ParseLengthOrPercentage(transforms[i + 3], 1);
```

```
03342                                 double d = ParseLengthOrPercentage(transforms[i + 4], 1);
03343                                 double e = ParseLengthOrPercentage(transforms[i + 5], 1);
03344                                 double f = ParseLengthOrPercentage(transforms[i + 6], 1);
03345 
03346                                 transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { a, c, e }, { b, d, f }, { 0, 0, 1 } });
03347 
03348                                 i += 6;
03349                             }
03350                             else if (transforms[i].Equals("translate",
      StringComparison.OrdinalIgnoreCase))
03351                             {
03352                                 double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03353 
03354                                 double y;
03355 
03356                                 if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03357                                 {
03358                                     transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03359                                     i += 2;
03360                                 }
03361                                 else
03362                                 {
03363                                     transformMatrix = MatrixUtils.Translate(transformMatrix, x, 0);
03364                                     i++;
03365                                 }
03366                             }
03367                             else if (transforms[i].Equals("scale",
      StringComparison.OrdinalIgnoreCase))
03368                             {
03369                                 double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03370 
03371                                 double y;
03372 
03373                                 if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03374                                 {
03375                                     transformMatrix = MatrixUtils.Scale(transformMatrix, x, y);
03376                                     i += 2;
03377                                 }
03378                                 else
03379                                 {
03380                                     transformMatrix = MatrixUtils.Scale(transformMatrix, x, x);
03381                                     i++;
03382                                 }
03383                             }
03384                             else if (transforms[i].Equals("rotate",
      StringComparison.OrdinalIgnoreCase))
03385                             {
03386                                 double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03387 
03388                                 double x, y;
03389 
03390                                 if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
03391                                 {
03392                                     transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03393                                     transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03394                                     transformMatrix = MatrixUtils.Translate(transformMatrix, -x, -y);
03395                                     i += 2;
03396                                 }
03397                                 else
03398                                 {
03399                                     transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03400                                     i++;
03401                                 }
03402                             }
03403                             else if (transforms[i].Equals("skewX",
      StringComparison.OrdinalIgnoreCase))
03404                             {
03405                                 double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03406 
03407                                 transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, Math.Tan(psi), 0 }, { 0, 1, 0 }, { 0, 0, 1 } });
03408                                 i++;
03409                             }
03410                             else if (transforms[i].Equals("skewY",
      StringComparison.OrdinalIgnoreCase))
03411                             {
03412                                 double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03413 
03414                                 transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
```

```
    { { 1, 0, 0 }, { Math.Tan(psi), 1, 0 }, { 0, 0, 1 } });
03415                                 i++;
03416                             }
03417                         }
03418
03419                         double determinant = transformMatrix[0, 0] * (transformMatrix[1, 1] *
    transformMatrix[2, 2] - transformMatrix[1, 2] * transformMatrix[2, 1]) -
03420                             transformMatrix[0, 1] * (transformMatrix[1, 0] * transformMatrix[2, 2] -
    transformMatrix[1, 2] * transformMatrix[2, 0]) +
03421                             transformMatrix[0, 2] * (transformMatrix[1, 0] * transformMatrix[2, 1] -
    transformMatrix[1, 1] * transformMatrix[2, 0]);
03422
03423                         double[] focus = MatrixUtils.Multiply(transformMatrix, new double[] { fx, fy
    });
03424                         double[] centre = MatrixUtils.Multiply(transformMatrix, new double[] { cx, cy
    });
03425
03426                         fx = focus[0];
03427                         fy = focus[1];
03428                         cx = centre[0];
03429                         cy = centre[1];
03430                         r = r * Math.Sqrt(determinant);
03431                     }
03432
03433                     tbr.Add(id, (userSpaceOnUse, new RadialGradientBrush(new Point(fx, fy), new
    Point(cx, cy), r, gradientStops)));
03434                 }
03435             }
03436
03437
03438             return tbr;
03439         }
03440
03441         private static KeyValuePair<string, FontFamily>?  ParseFontFaceBlock(List<string> tokens)
03442         {
03443             int fontFamilyInd = tokens.IndexOf("font-family");
03444             string fontFamilyName = tokens[fontFamilyInd + 2].Trim().Trim('"').Trim();
03445
03446             int srcInd = tokens.IndexOf("src");
03447             string src = tokens[srcInd + 2];
03448
03449             string mimeType = src.Substring(src.IndexOf("data:") + 5);
03450             mimeType = mimeType.Substring(0, mimeType.IndexOf(";"));
03451
03452             if (mimeType.Equals("font/ttf", StringComparison.OrdinalIgnoreCase) ||
    mimeType.Equals("font/truetype", StringComparison.OrdinalIgnoreCase) ||
    mimeType.Equals("application/x-font-ttf", StringComparison.OrdinalIgnoreCase))
03453             {
03454                 src = src.Substring(src.IndexOf("base64,") + 7);
03455                 src = src.TrimEnd(')').TrimEnd('\"').TrimEnd(')');
03456                 byte[] fontBytes = Convert.FromBase64String(src);
03457
03458                 string tempFile = Path.GetTempFileName();
03459
03460                 File.WriteAllBytes(tempFile, fontBytes);
03461
03462                 FontFamily family = FontFamily.ResolveFontFamily(tempFile);
03463                 return new KeyValuePair<string, FontFamily>(fontFamilyName, family);
03464             }
03465
03466             return null;
03467         }
03468
03469         private const string CSSDelimiters = ":;,{}";
03470
03471         private static string ReadCSSToken(StringReader reader)
03472         {
03473             StringBuilder tbr = new StringBuilder();
03474
03475             bool openQuotes = false;
03476             int openParentheses = 0;
03477
03478             int c = reader.Read();
03479             if (c >= 0)
03480             {
03481                 tbr.Append((char)c);
03482
03483                 if ((char)c == '"')
03484                 {
03485                     openQuotes = !openQuotes;
03486                 }
03487
03488                 if ((char)c == '(')
03489                 {
03490                     openParentheses++;
03491                 }
03492                 if ((char)c == ')')
```

```
03493                 {
03494                     openParentheses--;
03495                 }
03496 
03497 
03498                 while (c >= 0 && (!CSSDelimiters.Contains((char)c) || openQuotes || openParentheses >
      0))
03499                 {
03500                     c = reader.Read();
03501                     tbr.Append((char)c);
03502                     if ((char)c == '"')
03503                     {
03504                         openQuotes = !openQuotes;
03505                     }
03506                     if ((char)c == '(')
03507                     {
03508                         openParentheses++;
03509                     }
03510                     if ((char)c == ')')
03511                     {
03512                         openParentheses--;
03513                     }
03514                     c = reader.Peek();
03515                 }
03516             }
03517 
03518             string val = tbr.ToString().Trim();
03519 
03520             return (string.IsNullOrEmpty(val) && c >= 0) ?  ReadCSSToken(reader) :  val;
03521         }
03522     }
03523 
03524     internal class PathTransformerGraphicsContext :  IGraphicsContext
03525     {
03526         public GraphicsPath CurrentPath = new GraphicsPath();
03527         public double[,] TransformMatrix = MatrixUtils.Identity;
03528 
03529         private Stack<double[,]> transformMatrices;
03530         public PathTransformerGraphicsContext()
03531         {
03532             transformMatrices = new Stack<double[,]>();
03533             transformMatrices.Push((double[,])TransformMatrix.Clone());
03534         }
03535 
03536         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
03537         {
03538             double[] p1 = MatrixUtils.Multiply(TransformMatrix, new double[] { p1X, p1Y });
03539             double[] p2 = MatrixUtils.Multiply(TransformMatrix, new double[] { p2X, p2Y });
03540             double[] p3 = MatrixUtils.Multiply(TransformMatrix, new double[] { p3X, p3Y });
03541 
03542             CurrentPath.CubicBezierTo(p1[0], p1[1], p2[0], p2[1], p3[0], p3[1]);
03543         }
03544 
03545         public void LineTo(double x, double y)
03546         {
03547             double[] p = MatrixUtils.Multiply(TransformMatrix, new double[] { x, y });
03548             CurrentPath.LineTo(p[0], p[1]);
03549         }
03550 
03551         public void MoveTo(double x, double y)
03552         {
03553             double[] p = MatrixUtils.Multiply(TransformMatrix, new double[] { x, y });
03554             CurrentPath.MoveTo(p[0], p[1]);
03555         }
03556 
03557         public void Close()
03558         {
03559             CurrentPath.Close();
03560         }
03561 
03562         public void Restore()
03563         {
03564             TransformMatrix = transformMatrices.Pop();
03565         }
03566 
03567         public void Rotate(double angle)
03568         {
03569             TransformMatrix = MatrixUtils.Rotate(TransformMatrix, angle);
03570         }
03571 
03572         public void Save()
03573         {
03574             transformMatrices.Push((double[,])TransformMatrix.Clone());
03575         }
03576 
03577         public void Scale(double scaleX, double scaleY)
```

```
03578         {
03579             TransformMatrix = MatrixUtils.Scale(TransformMatrix, scaleX, scaleY);
03580         }
03581 
03582         public void Transform(double a, double b, double c, double d, double e, double f)
03583         {
03584             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
03585             TransformMatrix = MatrixUtils.Multiply(TransformMatrix, transfMatrix);
03586         }
03587 
03588         public void Translate(double x, double y)
03589         {
03590             TransformMatrix = MatrixUtils.Translate(TransformMatrix, x, y);
03591         }
03592 
03593 
03594 
03595         public double Width => throw new NotImplementedException();
03596 
03597         public double Height => throw new NotImplementedException();
03598 
03599         public Font Font { get => throw new NotImplementedException(); set => throw new
      NotImplementedException(); }
03600         public TextBaselines TextBaseline { get => throw new NotImplementedException(); set => throw
      new NotImplementedException(); }
03601 
03602         public Brush FillStyle => Colours.Black;
03603 
03604         public Brush StrokeStyle => Colours.Black;
03605 
03606         public double LineWidth { get => 1; set { } }
03607         public LineCaps LineCap { set { } }
03608         public LineJoins LineJoin { set { } }
03609         public string Tag { get => null; set { } }
03610 
03611         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
03612         {
03613             throw new NotImplementedException();
03614         }
03615 
03616         public void Fill(FillRule fillRule)
03617         {
03618 
03619         }
03620 
03621         public void FillText(string text, double x, double y)
03622         {
03623             throw new NotImplementedException();
03624         }
03625 
03626         public void Rectangle(double x0, double y0, double width, double height)
03627         {
03628             MoveTo(x0, y0);
03629             LineTo(x0 + width, y0);
03630             LineTo(x0 + width, y0 + height);
03631             LineTo(x0, y0 + height);
03632             Close();
03633         }
03634 
03635         public void SetClippingPath()
03636         {
03637             throw new NotImplementedException();
03638         }
03639 
03640         public void SetFillStyle((int r, int g, int b, double a) style)
03641         {
03642 
03643         }
03644 
03645         public void SetFillStyle(Brush style)
03646         {
03647 
03648         }
03649 
03650         public void SetLineDash(LineDash dash)
03651         {
03652 
03653         }
03654 
03655         public void SetStrokeStyle((int r, int g, int b, double a) style)
03656         {
03657 
03658         }
03659 
03660         public void SetStrokeStyle(Brush style)
```

```
03661         {
03662
03663         }
03664
03665         public void Stroke()
03666         {
03667
03668         }
03669
03670         public void StrokeText(string text, double x, double y)
03671         {
03672             throw new NotImplementedException();
03673         }
03674
03675         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
03676         {
03677             graphics.CopyToIGraphicsContext(this);
03678         }
03679     }
03680 }
```

8.46 Lights.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.ThreeD
00023 {
00024 /// <summary>
00025 /// Represents the intensity of a light source at a particular point.
00026 /// </summary>
00027     public struct LightIntensity
00028     {
00029 /// <summary>
00030 /// The intensity of the light.
00031 /// </summary>
00032         public double Intensity;
00033
00034 /// <summary>
00035 /// The direction towards from which the light comes.
00036 /// </summary>
00037         public NormalizedVector3D Direction;
00038
00039 /// <summary>
00040 /// Creates a new <see cref="LightIntensity"/>.
00041 /// </summary>
00042 /// <param name="intensity">The intensity of the light.</param>
00043 /// <param name="direction">The direction from which the light comes.</param>
00044         public LightIntensity(double intensity, NormalizedVector3D direction)
00045         {
00046             this.Intensity = intensity;
00047             this.Direction = direction;
00048         }
00049
00050 /// <summary>
00051 /// Deconstructs the struct.
00052 /// </summary>
00053 /// <param name="intensity">This parameter will hold the <see cref="Intensity"/> of the light.</param>
00054 /// <param name="direction">This parameter will hold the <see cref="Direction"/> of the light.</param>
00055         public void Deconstruct(out double intensity, out NormalizedVector3D direction)
00056         {
00057             intensity = this.Intensity;
00058             direction = this.Direction;
00059         }
00060     }
```

```
00061
00062 /// <summary>
00063 /// Represents a light source.
00064 /// </summary>
00065     public interface ILightSource
00066     {
00067 /// <summary>
00068 /// Computes the light intensity at the specified point, without taking into account any obstructions.
00069 /// </summary>
00070 /// <param name="point">The <see cref="Point3DElement"/> at which the light intensity should be
      computed.</param>
00071 /// <returns></returns>
00072         LightIntensity GetLightAt(Point3D point);
00073
00074 /// <summary>
00075 /// Determines whether the light casts a shadow or not.
00076 /// </summary>
00077         bool CastsShadow { get; }
00078
00079 /// <summary>
00080 /// Determines the amount of obstruction of the light that results at <paramref name="point"/> due to
      the specified <paramref name="shadowingTriangles"/>.
00081 /// </summary>
00082 /// <param name="point">The <see cref="Point3D"/> at which the obstruction should be computed.</param>
00083 /// <param name="shadowingTriangles">A collection of <see cref="Triangle3DElement"/> casting
      shadows.</param>
00084 /// <returns>1 if the light is completely obstructed, 0 if the light is completely visible, a value
      between these if the light is partially obstructed.</returns>
00085         double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles);
00086     }
00087
00088 /// <summary>
00089 /// Represents a uniform ambien light source.
00090 /// </summary>
00091     public class AmbientLightSource :  ILightSource
00092     {
00093 /// <summary>
00094 /// The intensity of the light.
00095 /// </summary>
00096         public double Intensity { get; set; }
00097
00098 /// <inheritdoc/>
00099         public bool CastsShadow => false;
00100
00101 /// <summary>
00102 /// Creates a new <see cref="AmbientLightSource"/> instance.
00103 /// </summary>
00104 /// <param name="intensity">The intensity of the light.</param>
00105         public AmbientLightSource(double intensity)
00106         {
00107             this.Intensity = intensity;
00108         }
00109
00110 /// <inheritdoc/>
00111         public LightIntensity GetLightAt(Point3D point)
00112         {
00113             return new LightIntensity(Intensity, new NormalizedVector3D(double.NaN, double.NaN,
      double.NaN));
00114         }
00115
00116 /// <inheritdoc/>
00117         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00118         {
00119             return 0;
00120         }
00121     }
00122
00123 /// <summary>
00124 /// Represents a parallel light source.
00125 /// </summary>
00126     public class ParallelLightSource :  ILightSource
00127     {
00128 /// <summary>
00129 /// The intensity of the light.
00130 /// </summary>
00131         public double Intensity { get; set; }
00132
00133 /// <summary>
00134 /// The direction along which the light travels.
00135 /// </summary>
00136         public NormalizedVector3D Direction { get; }
00137
00138 /// <summary>
00139 /// The reverse of <see cref="Direction"/>.
00140 /// </summary>
00141         public NormalizedVector3D ReverseDirection { get; }
00142
```

```
00143 /// <inheritdoc/>
00144         public bool CastsShadow { get; set; } = true;
00145 
00146 /// <summary>
00147 /// Creates a new <see cref="ParallelLightSource"/> instance.
00148 /// </summary>
00149 /// <param name="intensity">The intensity of the light.</param>
00150 /// <param name="direction">The direction along which the light travels.</param>
00151         public ParallelLightSource(double intensity, NormalizedVector3D direction)
00152         {
00153             this.Intensity = intensity;
00154             this.Direction = direction;
00155             this.ReverseDirection = direction.Reverse();
00156         }
00157 
00158 /// <inheritdoc/>
00159         public LightIntensity GetLightAt(Point3D point)
00160         {
00161             return new LightIntensity(Intensity, Direction);
00162         }
00163 
00164 /// <inheritdoc/>
00165         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00166         {
00167             foreach (Triangle3DElement triangle in shadowingTriangles)
00168             {
00169                 Point3D? projected = triangle.ProjectOnThisPlane(point, ReverseDirection, true,
    double.PositiveInfinity);
00170 
00171                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
    triangle.Point1, triangle.Point2, triangle.Point3))
00172                 {
00173                     return 1;
00174                 }
00175             }
00176 
00177             return 0;
00178         }
00179     }
00180 
00181 /// <summary>
00182 /// Represents a point light source.
00183 /// </summary>
00184     public class PointLightSource :  ILightSource
00185     {
00186 /// <inheritdoc/>
00187         public bool CastsShadow { get; set; } = true;
00188 
00189 /// <summary>
00190 /// The position of the light source.
00191 /// </summary>
00192         public Point3D Position { get; set; }
00193 
00194 /// <summary>
00195 /// The base intensity of the light.
00196 /// </summary>
00197         public double Intensity { get; set; }
00198 
00199 /// <summary>
00200 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
    disable distance attenuation.
00201 /// </summary>
00202         public double DistanceAttenuationExponent { get; set; } = 2;
00203 
00204 /// <summary>
00205 /// Creates a new <see cref="PointLightSource"/> instance.
00206 /// </summary>
00207 /// <param name="intensity">The intensity of the light.</param>
00208 /// <param name="position">The position of the light source.</param>
00209         public PointLightSource(double intensity, Point3D position)
00210         {
00211             this.Position = position;
00212             this.Intensity = intensity;
00213         }
00214 
00215 /// <inheritdoc/>
00216         public LightIntensity GetLightAt(Point3D point)
00217         {
00218             if (DistanceAttenuationExponent == 2)
00219             {
00220                 return new LightIntensity(Intensity / ((point.X - Position.X) * (point.X - Position.X)
    + (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z)),
    (point - Position).Normalize());
00221             }
00222             else if (DistanceAttenuationExponent == 0)
00223             {
00224                 return new LightIntensity(Intensity, (point - Position).Normalize());
```

```
00225             }
00226             else
00227             {
00228                 return new LightIntensity(Intensity / Math.Pow((point.X - Position.X) * (point.X - Position.X) + (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z), DistanceAttenuationExponent * 0.5), (point - Position).Normalize());
00229             }
00230         }
00231 
00232 /// <inheritdoc/>
00233         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00234         {
00235             Vector3D reverseDir = this.Position - point;
00236             double maxD = reverseDir.Modulus;
00237             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD, reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00238 
00239             foreach (Triangle3DElement triangle in shadowingTriangles)
00240             {
00241                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00242 
00243                 if (projected != null && Intersections3D.PointInTriangle(projected.Value, triangle.Point1, triangle.Point2, triangle.Point3))
00244                 {
00245                     return 1;
00246                 }
00247             }
00248 
00249             return 0;
00250         }
00251     }
00252 
00253 /// <summary>
00254 /// Represents a conic spotlight.
00255 /// </summary>
00256     public class SpotlightLightSource :  ILightSource
00257     {
00258 /// <inheritdoc/>
00259         public bool CastsShadow { get; set; } = true;
00260 
00261 /// <summary>
00262 /// The position of the light source.
00263 /// </summary>
00264         public Point3D Position { get; set; }
00265 
00266 /// <summary>
00267 /// The direction of the cone axis.
00268 /// </summary>
00269         public NormalizedVector3D Direction { get; set; }
00270 
00271 /// <summary>
00272 /// The base intensity of the light.
00273 /// </summary>
00274         public double Intensity { get; set; }
00275 
00276 /// <summary>
00277 /// The angular size of the light cone, in radians.
00278 /// </summary>
00279         public double BeamWidthAngle { get; set; }
00280 
00281 /// <summary>
00282 /// The angular size of the cutoff cone, in radians.
00283 /// </summary>
00284         public double CutoffAngle { get; set; }
00285 
00286 /// <summary>
00287 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to disable distance attenuation.
00288 /// </summary>
00289         public double DistanceAttenuationExponent { get; set; } = 2;
00290 
00291 /// <summary>
00292 /// An exponent determining how fast the light attenuates between the main light cone and the cutoff cone.
00293 /// </summary>
00294         public double AngleAttenuationExponent { get; set; } = 1;
00295 
00296 /// <summary>
00297 /// Creates a new <see cref="SpotlightLightSource"/> instance.
00298 /// </summary>
00299 /// <param name="intensity">The intensity of the light.</param>
00300 /// <param name="position">The position of the light source.</param>
00301 /// <param name="direction">The direction of the cone's axis.</param>
00302 /// <param name="beamWidthAngle">The angular size of the light cone, in radians.</param>
00303 /// <param name="cutoffAngle">The angular size of the cutoff cone, in radians.</param>
00304         public SpotlightLightSource(double intensity, Point3D position, NormalizedVector3D direction, double beamWidthAngle, double cutoffAngle)
```

```
00305         {
00306             this.Position = position;
00307             this.Direction = direction;
00308             this.Intensity = intensity;
00309             this.BeamWidthAngle = beamWidthAngle;
00310             this.CutoffAngle = cutoffAngle;
00311         }
00312 
00313 /// <inheritdoc/>
00314         public LightIntensity GetLightAt(Point3D point)
00315         {
00316             double angle = Math.Atan2(((point - Position) ^ Direction).Modulus, (point - Position) *
      Direction);
00317             if (angle > Math.PI)
00318             {
00319                 angle -= 2 * Math.PI;
00320             }
00321             angle = Math.Abs(angle);
00322             double intensity;
00323 
00324             if (DistanceAttenuationExponent == 0)
00325             {
00326                 intensity = Intensity;
00327             }
00328             else if (DistanceAttenuationExponent == 2)
00329             {
00330                 intensity = Intensity / ((point.X - Position.X) * (point.X - Position.X) + (point.Y -
      Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z));
00331             }
00332             else
00333             {
00334                 intensity = Intensity / Math.Pow((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z), 0.5
      * DistanceAttenuationExponent);
00335             }
00336 
00337             if (angle <= BeamWidthAngle)
00338             {
00339                 // * 1
00340             }
00341             else if (angle >= CutoffAngle)
00342             {
00343                 intensity = 0;
00344             }
00345             else if (AngleAttenuationExponent == 0)
00346             {
00347                 // * 1
00348             }
00349             else if (AngleAttenuationExponent == 1)
00350             {
00351                 intensity *= (CutoffAngle - angle) / (CutoffAngle - BeamWidthAngle);
00352             }
00353             else
00354             {
00355                 intensity *= Math.Pow((CutoffAngle - angle) / (CutoffAngle - BeamWidthAngle),
      AngleAttenuationExponent);
00356             }
00357 
00358             return new LightIntensity(intensity, (point - Position).Normalize());
00359         }
00360 
00361 /// <inheritdoc/>
00362         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00363         {
00364             Vector3D reverseDir = this.Position - point;
00365             double maxD = reverseDir.Modulus;
00366             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00367 
00368             foreach (Triangle3DElement triangle in shadowingTriangles)
00369             {
00370                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00371 
00372                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00373                 {
00374                     return 1;
00375                 }
00376             }
00377 
00378             return 0;
00379         }
00380     }
00381 
00382 /// <summary>
00383 /// Represents a point light source with a stencil in front of it.
00384 /// </summary>
```

```
00385     public class MaskedLightSource :  ILightSource
00386     {
00387 /// <inheritdoc/>
00388         public bool CastsShadow { get; set; } = true;
00389 
00390 /// <summary>
00391 /// The position of the light source.
00392 /// </summary>
00393         public Point3D Position { get; }
00394 
00395 /// <summary>
00396 /// The projection of the <see cref="Position"/> on the mask plane along the light's <see
      cref="Direction"/>.
00397 /// </summary>
00398         public Point3D Origin { get; }
00399 
00400 /// <summary>
00401 /// The direction of the light.
00402 /// </summary>
00403         public NormalizedVector3D Direction { get; }
00404 
00405 /// <summary>
00406 /// The distance between the light source and the mask plane.
00407 /// </summary>
00408         public double Distance { get; }
00409 
00410         private List<Point3D[]> TriangulatedMask { get; }
00411 
00412 /// <summary>
00413 /// The base intensity of the light.
00414 /// </summary>
00415         public double Intensity { get; set; }
00416 
00417 /// <summary>
00418 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
00419 /// </summary>
00420         public double DistanceAttenuationExponent { get; set; } = 2;
00421 
00422 /// <summary>
00423 /// An exponent determining how fast the light attenuates away from the light's axis.  Set to 0 to
      disable angular attenuation.
00424 /// </summary>
00425         public double AngleAttenuationExponent { get; set; } = 1;
00426 
00427 /// <summary>
00428 /// Creates a new <see cref="MaskedLightSource"/> by triangulating the specified <see
      cref="GraphicsPath"/>.
00429 /// </summary>
00430 /// <param name="intensity">The base intensity of the light.</param>
00431 /// <param name="position">The position of the light source.</param>
00432 /// <param name="direction">The direction of the light.</param>
00433 /// <param name="distance">The distance between the light source and the mask plane.</param>
00434 /// <param name="mask">A <see cref="GraphicsPath"/> representing the transparent part of the
      mask.</param>
00435 /// <param name="maskOrientation">An angle in radians determining the orientation of the 2D mask in
      the mask plane.</param>
00436 /// <param name="triangulationResolution">The resolution to use to triangulate the <paramref
      name="mask"/>.</param>
00437         public MaskedLightSource(double intensity, Point3D position, NormalizedVector3D direction,
      double distance, GraphicsPath mask, double maskOrientation, double triangulationResolution) :
      this(intensity, position, direction, distance, mask.Triangulate(triangulationResolution, true),
      maskOrientation)
00438         {
00439 
00440         }
00441 
00442 /// <summary>
00443 /// Creates a new <see cref="MaskedLightSource"/> using the specified <paramref
      name="triangulatedMask"/>.
00444 /// </summary>
00445 /// <param name="intensity">The base intensity of the light.</param>
00446 /// <param name="position">The position of the light source.</param>
00447 /// <param name="direction">The direction of the light.</param>
00448 /// <param name="distance">The distance between the light source and the mask plane.</param>
00449 /// <param name="triangulatedMask">A collection of <see cref="GraphicsPath"/>s representing the
      transparent part of the mask.  Each <see cref="GraphicsPath"/> should represent a single
      triangle.</param>
00450 /// <param name="maskOrientation">An angle in radians determining the orientation of the 2D mask in
      the mask plane.</param>
00451         public MaskedLightSource(double intensity, Point3D position, NormalizedVector3D direction,
      double distance, IEnumerable<GraphicsPath> triangulatedMask, double maskOrientation)
00452         {
00453             this.Intensity = intensity;
00454             this.Position = position;
00455             this.Direction = direction;
00456             this.Distance = distance;
```

```
00457             this.Origin = this.Position + distance * this.Direction;
00458
00459             double[,] rotation = Matrix3D.RotationToAlignWithZ(direction).Inverse();
00460
00461             double[,] rotation2 = Matrix3D.RotationAroundAxis(direction, maskOrientation);
00462
00463             List<Point3D[]> maskList = new List<Point3D[]>();
00464
00465             foreach (GraphicsPath trianglePath in triangulatedMask)
00466             {
00467                 Point3D[] triangle = new Point3D[3];
00468
00469                 List<Point> points = trianglePath.GetPoints().First();
00470
00471                 for (int i = 0; i < 3; i++)
00472                 {
00473                     triangle[i] = (Point3D)((Vector3D)(rotation2 * (rotation * new
      Point3D(points[i].X, points[i].Y, 0))) + Origin);
00474                 }
00475
00476                 maskList.Add(triangle);
00477             }
00478
00479             this.TriangulatedMask = maskList;
00480         }
00481
00482 /// <inheritdoc/>
00483         public LightIntensity GetLightAt(Point3D point)
00484         {
00485             double d = Distance / ((point - this.Position) * this.Direction);
00486             Point3D pt = this.Position + (point - this.Position) * d;
00487
00488             bool contained = false;
00489
00490             foreach (Point3D[] triangle in TriangulatedMask)
00491             {
00492                 if (Intersections3D.PointInTriangle(pt, triangle[0], triangle[1], triangle[2]))
00493                 {
00494                     contained = true;
00495                     break;
00496                 }
00497             }
00498
00499             if (contained)
00500             {
00501                 double intensity;
00502
00503                 if (DistanceAttenuationExponent == 0)
00504                 {
00505                     intensity = Intensity;
00506                 }
00507                 else if (DistanceAttenuationExponent == 2)
00508                 {
00509                     intensity = Intensity / ((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z));
00510                 }
00511                 else
00512                 {
00513                     intensity = Intensity / Math.Pow((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z), 0.5
      * DistanceAttenuationExponent);
00514                 }
00515
00516                 if (AngleAttenuationExponent == 0)
00517                 {
00518                     // * 1
00519                 }
00520                 else
00521                 {
00522                     double angle = Math.Atan2(((point - Position) ^ Direction).Modulus, (point -
      Position) * Direction);
00523                     if (angle > Math.PI)
00524                     {
00525                         angle -= 2 * Math.PI;
00526                     }
00527
00528                     if (Math.Abs(angle) < Math.PI / 2)
00529                     {
00530                         angle = Math.Abs(angle) / Math.PI * 2;
00531
00532                         if (AngleAttenuationExponent == 1)
00533                         {
00534                             intensity *= 1 - angle;
00535                         }
00536                         else
00537                         {
00538                             intensity *= Math.Pow(1 - angle, AngleAttenuationExponent);
```

```
00539                         }
00540                     }
00541                     else
00542                     {
00543                         intensity = 0;
00544                     }
00545                 }
00546 
00547                 return new LightIntensity(intensity, (point - Position).Normalize());
00548             }
00549             else
00550             {
00551                 return new LightIntensity(0, (point - Position).Normalize());
00552             }
00553         }
00554 
00555 /// <inheritdoc/>
00556         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00557         {
00558             Vector3D reverseDir = this.Position - point;
00559             double maxD = reverseDir.Modulus;
00560             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00561 
00562             foreach (Triangle3DElement triangle in shadowingTriangles)
00563             {
00564                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00565 
00566                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00567                 {
00568                     return 1;
00569                 }
00570             }
00571 
00572             return 0;
00573         }
00574     }
00575 
00576 /// <summary>
00577 /// Represents a light source emitting light from a circular area.
00578 /// </summary>
00579     public class AreaLightSource :  ILightSource
00580     {
00581 /// <inheritdoc/>
00582         public bool CastsShadow { get; set; } = true;
00583 
00584 /// <summary>
00585 /// The centre of the light-emitting area.
00586 /// </summary>
00587         public Point3D Center { get; }
00588 
00589         private Point3D VirtualSource { get; }
00590 
00591 /// <summary>
00592 /// The direction of the light's main axis, i.e.  the normal to the plane containing the
      light-emitting area.
00593 /// </summary>
00594         public NormalizedVector3D Direction { get; }
00595 
00596 /// <summary>
00597 /// The radius of the light emitting area.
00598 /// </summary>
00599         public double Radius { get; }
00600 
00601 /// <summary>
00602 /// The radius of the penumbra area.
00603 /// </summary>
00604         public double PenumbraRadius { get; }
00605 
00606 /// <summary>
00607 /// The base intensity of the light.
00608 /// </summary>
00609         public double Intensity { get; set; }
00610 
00611 /// <summary>
00612 /// The distance between the focal point of the light and the light's <see cref="Center"/>.
00613 /// </summary>
00614         public double SourceDistance { get; }
00615 
00616 /// <summary>
00617 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
00618 /// </summary>
00619         public double DistanceAttenuationExponent { get; set; } = 2;
00620 
00621 /// <summary>
```

```
00622 /// An exponent determining how fast the light attenuates between the light-emitting area radius and
      the penumbra radius.
00623 /// </summary>
00624         public double PenumbraAttenuationExponent { get; set; } = 1;
00625
00626 /// <summary>
00627 /// The number of points to use when determining the amount of light that is obstructed at a certain
      point.
00628 /// </summary>
00629         public int ShadowSamplingPointCount { get; }
00630
00631         private Point3D[] ShadowSamplingPoints { get; }
00632
00633 /// <summary>
00634 /// Creates a new <see cref="AreaLightSource"/> instance.
00635 /// </summary>
00636 /// <param name="intensity">The base intensity of the light.</param>
00637 /// <param name="center">The centre of the light-emitting area.</param>
00638 /// <param name="radius">The radius of the light-emitting area.</param>
00639 /// <param name="penumbraRadius">The radius of the penumbra area.</param>
00640 /// <param name="direction">The direction of the light.</param>
00641 /// <param name="sourceDistance">The distance between the focal point of the light and the light's
      center.</param>
00642 /// <param name="shadowSamplingPointCount">The number of points to use when determining the amount of
      light that is obstructed at a certain point.</param>
00643         public AreaLightSource(double intensity, Point3D center, double radius, double penumbraRadius,
      NormalizedVector3D direction, double sourceDistance, int shadowSamplingPointCount)
00644         {
00645             if (shadowSamplingPointCount < 1)
00646             {
00647                 throw new ArgumentOutOfRangeException(nameof(shadowSamplingPointCount),
      shadowSamplingPointCount, "The number of shadow sampling points must be greater than or equal to 1!");
00648             }
00649
00650             this.Center = center;
00651             this.Direction = direction;
00652             this.Intensity = intensity;
00653             this.Radius = radius;
00654             this.PenumbraRadius = penumbraRadius;
00655             this.SourceDistance = sourceDistance;
00656
00657             this.VirtualSource = this.Center - this.Direction * this.SourceDistance;
00658
00659             this.ShadowSamplingPointCount = shadowSamplingPointCount;
00660
00661             NormalizedVector3D yAxis;
00662
00663             if (Math.Abs(new Vector3D(0, 1, 0) * this.Direction) < 1)
00664             {
00665                 yAxis = (new Vector3D(0, 1, 0) - (new Vector3D(0, 1, 0) * this.Direction) *
      this.Direction).Normalize();
00666             }
00667             else
00668             {
00669                 yAxis = (new Vector3D(0, 0, 1) - (new Vector3D(0, 0, 1) * this.Direction) *
      this.Direction).Normalize();
00670             }
00671
00672             NormalizedVector3D xAxis = (this.Direction ^ yAxis).Normalize();
00673
00674             this.ShadowSamplingPoints = new Point3D[shadowSamplingPointCount - 1];
00675
00676             for (int i = 0; i < shadowSamplingPointCount - 1; i++)
00677             {
00678                 double r = ((double)i / (shadowSamplingPointCount - 2)) * this.Radius;
00679                 double theta = (double)i / (shadowSamplingPointCount - 2) * 2 * Math.PI *
      ((shadowSamplingPointCount - 1) / 3.7);
00680
00681                 double x = r * Math.Cos(theta);
00682                 double y = r * Math.Sin(theta);
00683
00684                 Point3D pt = this.Center + x * xAxis + y * yAxis;
00685                 this.ShadowSamplingPoints[i] = pt;
00686             }
00687         }
00688
00689 /// <inheritdoc/>
00690         public LightIntensity GetLightAt(Point3D point)
00691         {
00692             double denom = ((point - this.VirtualSource) * this.Direction);
00693
00694             if (denom <= 0)
00695             {
00696                 return new LightIntensity(0, (point - this.VirtualSource).Normalize());
00697             }
00698             else
00699             {
```

```
00700                 double d = this.SourceDistance / denom;
00701                 Point3D pt = this.VirtualSource + d * (point - this.VirtualSource);
00702                 double distSq = (pt.X - this.Center.X) * (pt.X - this.Center.X) + (pt.Y -
      this.Center.Y) * (pt.Y - this.Center.Y) + (pt.Z - this.Center.Z) * (pt.Z - this.Center.Z);
00703 
00704                 if (distSq < this.Radius * this.Radius)
00705                 {
00706                     double intensity;
00707                     if (DistanceAttenuationExponent == 0)
00708                     {
00709                         intensity = Intensity;
00710                     }
00711                     else if (DistanceAttenuationExponent == 2)
00712                     {
00713                         intensity = Intensity / ((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z));
00714                     }
00715                     else
00716                     {
00717                         intensity = Intensity / Math.Pow((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z), 0.5 * DistanceAttenuationExponent);
00718                     }
00719 
00720                     return new LightIntensity(intensity, (point - this.VirtualSource).Normalize());
00721                 }
00722                 else if (distSq < this.PenumbraRadius * this.PenumbraRadius)
00723                 {
00724                     double intensity;
00725                     if (DistanceAttenuationExponent == 0)
00726                     {
00727                         intensity = Intensity;
00728                     }
00729                     else if (DistanceAttenuationExponent == 2)
00730                     {
00731                         intensity = Intensity / ((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - VirtualSource.Z) * (point.Z - this.VirtualSource.Z));
00732                     }
00733                     else
00734                     {
00735                         intensity = Intensity / Math.Pow((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z), 0.5 * DistanceAttenuationExponent);
00736                     }
00737 
00738                     if (PenumbraAttenuationExponent == 0)
00739                     {
00740                         //intensity *= 1;
00741                     }
00742                     else if (PenumbraAttenuationExponent == 1)
00743                     {
00744                         double factor = (Math.Sqrt(distSq) - this.Radius) / (this.PenumbraRadius -
      this.Radius);
00745                         intensity *= 1 - factor;
00746                     }
00747                     else
00748                     {
00749                         double factor = (Math.Sqrt(distSq) - this.Radius) / (this.PenumbraRadius -
      this.Radius);
00750                         intensity *= Math.Pow(1 - factor, PenumbraAttenuationExponent);
00751                     }
00752 
00753                     return new LightIntensity(intensity, (point - this.VirtualSource).Normalize());
00754                 }
00755                 else
00756                 {
00757                     return new LightIntensity(0, (point - this.VirtualSource).Normalize());
00758                 }
00759             }
00760         }
00761 
00762 /// <inheritdoc/>
00763         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00764         {
00765             double totalObstruction = 0;
00766             int sampleCount = 0;
00767 
00768             {
00769                 Vector3D reverseDir = this.VirtualSource - point;
00770 
00771                 double denom = (reverseDir * this.Direction);
00772 
00773                 if (denom == 0)
00774                 {
00775                     //totalObstruction += 0;
```

```
00776                     sampleCount++;
00777                 }
00778                 else
00779                 {
00780                     double d = this.SourceDistance / denom;
00781                     Point3D pt = this.VirtualSource + d * reverseDir;
00782 
00783                     double maxD = (point - pt).Modulus;
00784                     NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00785 
00786                     bool found = false;
00787 
00788                     foreach (Triangle3DElement triangle in shadowingTriangles)
00789                     {
00790                         Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true,
      maxD);
00791 
00792                         if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00793                         {
00794                             totalObstruction += 1;
00795                             sampleCount++;
00796                             found = true;
00797                             break;
00798                         }
00799                     }
00800 
00801                     if (!found)
00802                     {
00803                         //totalObstruction += 0;
00804                         sampleCount++;
00805                     }
00806                 }
00807             }
00808 
00809 
00810             for (int i = 0; i < ShadowSamplingPoints.Length; i++)
00811             {
00812                 Vector3D reverseDir = ShadowSamplingPoints[i] - point;
00813 
00814                 double maxD = reverseDir.Modulus;
00815                 NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00816 
00817                 bool found = false;
00818 
00819                 foreach (Triangle3DElement triangle in shadowingTriangles)
00820                 {
00821                     Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true,
      maxD);
00822 
00823                     if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00824                     {
00825                         totalObstruction += 1;
00826                         sampleCount++;
00827                         found = true;
00828                         break;
00829                     }
00830                 }
00831 
00832                 if (!found)
00833                 {
00834                     //totalObstruction += 0;
00835                     sampleCount++;
00836                 }
00837             }
00838 
00839             return totalObstruction / sampleCount;
00840         }
00841     }
00842 }
```

8.47 Materials.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
```

```
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics;
00021 using System.Linq;
00022 using System.Runtime.InteropServices.ComTypes;
00023 using System.Security.Cryptography.X509Certificates;
00024 using System.Text;
00025
00026 namespace VectSharp.ThreeD
00027 {
00028 /// <summary>
00029 /// Represents a material used to the determine the appearance of <see cref="Triangle3DElement"/>.
00030 /// </summary>
00031     public interface IMaterial
00032     {
00033 /// <summary>
00034 /// Obtains the <see cref="Colour"/> at the specified point.
00035 /// </summary>
00036 /// <param name="point">The point whose colour should be determined.</param>
00037 /// <param name="surfaceNormal">The normal to the surface at the specified <paramref
      name="point"/>.</param>
00038 /// <param name="camera">The camera being used to render the scene.</param>
00039 /// <param name="lights">A list of light sources that are present in the scene.</param>
00040 /// <param name="obstructions">A list of values indicating how obstructed each light source
      is.</param>
00041 /// <returns>The <see cref="Colour"/> of the specified point.</returns>
00042         Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions);
00043     }
00044
00045 /// <summary>
00046 /// Represents a material that always has the same colour, regardless of light.
00047 /// </summary>
00048     public class ColourMaterial :  IMaterial
00049     {
00050 /// <summary>
00051 /// The colour of the material.
00052 /// </summary>
00053         public Colour Colour { get; }
00054
00055 /// <summary>
00056 /// Creates a new <see cref="ColourMaterial"/> instance.
00057 /// </summary>
00058 /// <param name="colour">The colour of the material.</param>
00059         public ColourMaterial(Colour colour)
00060         {
00061             this.Colour = colour;
00062         }
00063
00064 /// <inheritdoc/>
00065         public Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions)
00066         {
00067             return Colour;
00068         }
00069     }
00070
00071 /// <summary>
00072 /// Represents a material that uses a Phong reflection model to determine the colour of the material
      based on the light sources that hit it.
00073 /// </summary>
00074     public class PhongMaterial :  IMaterial
00075     {
00076 /// <summary>
00077 /// The base colour of the material.
00078 /// </summary>
00079         public Colour Colour { get; }
00080
00081         private (double L, double a, double b) LabColour;
00082         private (double H, double S, double L) LabBlackHSL;
00083         private (double H, double S, double L) LabWhiteHSL;
00084         private double IntensityExponent;
00085         private double TotalLength;
00086
00087 /// <summary>
00088 /// A coefficient determining how much ambient light is reflected by the material.
00089 /// </summary>
```

```
00090         public double AmbientReflectionCoefficient { get; set; } = 1;
00091
00092 /// <summary>
00093 /// A coefficient determining how much directional light is reflected by the material.
00094 /// </summary>
00095         public double DiffuseReflectionCoefficient { get; set; } = 1;
00096
00097 /// <summary>
00098 /// A coefficient determining the intensity of specular highlights.
00099 /// </summary>
00100         public double SpecularReflectionCoefficient { get; set; } = 1;
00101
00102 /// <summary>
00103 /// A coefficient determining the extent of specular highlights.
00104 /// </summary>
00105         public double SpecularShininess { get; set; } = 1;
00106
00107 /// <summary>
00108 /// Creates a new <see cref="PhongMaterial"/> instance.
00109 /// </summary>
00110 /// <param name="colour">The base colour of the material.</param>
00111         public PhongMaterial(Colour colour)
00112         {
00113             this.Colour = colour;
00114             this.LabColour = colour.ToLab();
00115             this.LabBlackHSL = Colour.FromLab(0, LabColour.a, LabColour.b).ToHSL();
00116             this.LabWhiteHSL = Colour.FromLab(1, LabColour.a, LabColour.b).ToHSL();
00117             this.TotalLength = 1 + LabBlackHSL.L + (1 - LabWhiteHSL.L);
00118             this.IntensityExponent = Math.Log((LabBlackHSL.L + LabColour.L) / TotalLength) /
      Math.Log(0.5);
00119         }
00120
00121         private Colour GetScaledColour(double intensity)
00122         {
00123             intensity = Math.Max(Math.Min(intensity, 1), 0);
00124
00125             double pos = Math.Pow(intensity, IntensityExponent) * TotalLength;
00126
00127             if (pos <= LabBlackHSL.L)
00128             {
00129                 return Colour.FromHSL(LabBlackHSL.H, LabBlackHSL.S, pos).WithAlpha(Colour.A);
00130             }
00131             else if (pos >= 1 + LabBlackHSL.L)
00132             {
00133                 return Colour.FromHSL(LabWhiteHSL.H, LabWhiteHSL.S, LabWhiteHSL.L + pos - 1 -
      LabBlackHSL.L).WithAlpha(Colour.A);
00134             }
00135             else
00136             {
00137                 return Colour.FromLab(pos - LabBlackHSL.L, LabColour.a,
      LabColour.b).WithAlpha(Colour.A);
00138             }
00139         }
00140
00141 /// <inheritdoc/>
00142         public Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions)
00143         {
00144             double intensity = 0;
00145
00146             for (int i = 0; i < lights.Count; i++)
00147             {
00148                 (double lightIntensity, NormalizedVector3D lightDirection) =
      lights[i].GetLightAt(point);
00149                 lightIntensity *= 1 - obstructions[i];
00150
00151                 if (double.IsNaN(lightDirection.X))
00152                 {
00153                     intensity += lightIntensity * AmbientReflectionCoefficient;
00154                 }
00155                 else
00156                 {
00157                     double dotProd = lightDirection * surfaceNormal;
00158
00159                     intensity += lightIntensity * Math.Max(0, dotProd) * DiffuseReflectionCoefficient;
00160
00161                     if (dotProd > 0)
00162                     {
00163                         NormalizedVector3D mirroredDirection = (lightDirection - 2 * dotProd *
      surfaceNormal).Normalize();
00164
00165                         NormalizedVector3D cameraDirection = (camera.ViewPoint - point).Normalize();
00166
00167                         double dotProd2 = mirroredDirection * cameraDirection;
00168
00169                         if (dotProd2 >= 0)
00170                         {
```

```
00171                             intensity += lightIntensity * Math.Pow(dotProd2, SpecularShininess) *
      SpecularReflectionCoefficient;
00172                         }
00173                     }
00174                 }
00175             }
00176
00177             return GetScaledColour(intensity);
00178         }
00179     }
00180 }
```

8.48 ObjectFactory.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.Text;
00022
00023 namespace VectSharp.ThreeD
00024 {
00025 /// <summary>
00026 /// A static class containing methods to create complex 3D objects.
00027 /// </summary>
00028     public static class ObjectFactory
00029     {
00030 /// <summary>
00031 /// Creates a cube.
00032 /// </summary>
00033 /// <param name="center">The centre of the cube.</param>
00034 /// <param name="size">The length of each side of the cube.</param>
00035 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00036 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00037 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00038 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the cube.</returns>
00039         public static List<Element3D> CreateCube(Point3D center, double size, IEnumerable<IMaterial>
      fill, string tag = null, int zIndex = 0)
00040         {
00041             return CreateCuboid(center, size, size, size, fill, tag, zIndex);
00042         }
00043
00044 /// <summary>
00045 /// Creates a cuboid.
00046 /// </summary>
00047 /// <param name="center">The centre of the cube.</param>
00048 /// <param name="sizeX">The length of the sides of the cube parallel to the x axis.</param>
00049 /// <param name="sizeY">The length of the sides of the cube parallel to the y axis.</param>
00050 /// <param name="sizeZ">The length of the sides of the cube parallel to the z axis.</param>
00051 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00052 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00053 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00054 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the cuboid.</returns>
00055         public static List<Element3D> CreateCuboid(Point3D center, double sizeX, double sizeY, double
      sizeZ, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00056         {
00057             Point3D A = new Point3D(center.X - sizeX * 0.5, center.Y - sizeY * 0.5, center.Z - sizeZ *
      0.5);
00058             Point3D B = new Point3D(center.X + sizeX * 0.5, center.Y - sizeY * 0.5, center.Z - sizeZ *
      0.5);
00059             Point3D C = new Point3D(center.X + sizeX * 0.5, center.Y + sizeY * 0.5, center.Z - sizeZ *
      0.5);
```

```
00060             Point3D D = new Point3D(center.X - sizeX * 0.5, center.Y + sizeY * 0.5, center.Z - sizeZ *
      0.5);
00061
00062             Point3D E = new Point3D(center.X - sizeX * 0.5, center.Y - sizeY * 0.5, center.Z + sizeZ *
      0.5);
00063             Point3D F = new Point3D(center.X + sizeX * 0.5, center.Y - sizeY * 0.5, center.Z + sizeZ *
      0.5);
00064             Point3D G = new Point3D(center.X + sizeX * 0.5, center.Y + sizeY * 0.5, center.Z + sizeZ *
      0.5);
00065             Point3D H = new Point3D(center.X - sizeX * 0.5, center.Y + sizeY * 0.5, center.Z + sizeZ *
      0.5);
00066
00067
00068             List<Element3D> tbr = new List<Element3D>();
00069
00070             tbr.AddRange(CreateRectangle(A, B, C, D, fill, tag, zIndex));
00071             tbr.AddRange(CreateRectangle(H, G, F, E, fill, tag, zIndex));
00072
00073             tbr.AddRange(CreateRectangle(E, F, B, A, fill, tag, zIndex));
00074             tbr.AddRange(CreateRectangle(G, H, D, C, fill, tag, zIndex));
00075
00076             tbr.AddRange(CreateRectangle(A, D, H, E, fill, tag, zIndex));
00077             tbr.AddRange(CreateRectangle(F, G, C, B, fill, tag, zIndex));
00078
00079             return tbr;
00080         }
00081
00082 /// <summary>
00083 /// Creates a quadrilater.  All the vertices need not be coplanar.
00084 /// </summary>
00085 /// <param name="point1">The first vertex of the quadrilater.</param>
00086 /// <param name="point2">The second vertex of the quadrilater.</param>
00087 /// <param name="point3">The third vertex of the quadrilater.</param>
00088 /// <param name="point4">The fourth vertex of the quadrilater.</param>
00089 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00090 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00091 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00092 /// <returns>A list containing two <see cref="Triangle3DElement"/>s representing the
      quadrilater.</returns>
00093         public static List<Element3D> CreateRectangle(Point3D point1, Point3D point2, Point3D point3,
      Point3D point4, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00094         {
00095             Triangle3DElement triangle1 = new Triangle3DElement(point1, point2, point3);
00096             triangle1.Fill.AddRange(fill);
00097             triangle1.Tag = tag;
00098             triangle1.ZIndex = zIndex;
00099
00100             Triangle3DElement triangle2 = new Triangle3DElement(point1, point3, point4);
00101             triangle2.Fill.AddRange(fill);
00102             triangle2.Tag = tag;
00103             triangle2.ZIndex = zIndex;
00104
00105             return new List<Element3D> { triangle1, triangle2 };
00106         }
00107
00108 /// <summary>
00109 /// Creates a quadrilater, specifying the vertex normals at the four vertices.  All the vertices need
      not be coplanar.
00110 /// </summary>
00111 /// <param name="point1">The first vertex of the quadrilater.</param>
00112 /// <param name="point2">The second vertex of the quadrilater.</param>
00113 /// <param name="point3">The third vertex of the quadrilater.</param>
00114 /// <param name="point4">The fourth vertex of the quadrilater.</param>
00115 /// <param name="point1Normal">The vertex normal at the first vertex of the quadrilater.</param>
00116 /// <param name="point2Normal">The vertex normal at the second vertex of the quadrilater.</param>
00117 /// <param name="point3Normal">The vertex normal at the third vertex of the quadrilater.</param>
00118 /// <param name="point4Normal">The vertex normal at the fourth vertex of the quadrilater.</param>
00119 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00120 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00121 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00122 /// <returns>A list containing two <see cref="Triangle3DElement"/>s representing the
      quadrilater.</returns>
00123         public static List<Element3D> CreateRectangle(Point3D point1, Point3D point2, Point3D point3,
      Point3D point4, NormalizedVector3D point1Normal, NormalizedVector3D point2Normal, NormalizedVector3D
      point3Normal, NormalizedVector3D point4Normal, IEnumerable<IMaterial> fill, string tag = null, int
      zIndex = 0)
00124         {
00125             Triangle3DElement triangle1 = new Triangle3DElement(point1, point2, point3, point1Normal,
      point2Normal, point3Normal);
00126             triangle1.Fill.AddRange(fill);
00127             triangle1.Tag = tag;
```

```
00128             triangle1.ZIndex = zIndex;
00129
00130             Triangle3DElement triangle2 = new Triangle3DElement(point1, point3, point4, point1Normal,
      point3Normal, point4Normal);
00131             triangle2.Fill.AddRange(fill);
00132             triangle2.Tag = tag;
00133             triangle2.ZIndex = zIndex;
00134
00135             return new List<Element3D> { triangle1, triangle2 };
00136         }
00137
00138 /// <summary>
00139 /// Creates a sphere.
00140 /// </summary>
00141 /// <param name="center">The centre of the sphere.</param>
00142 /// <param name="radius">The radius of the sphere.</param>
00143 /// <param name="steps">The number of meridians and parallels to use when generating the
      sphere.</param>
00144 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00145 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00146 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00147 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the sphere.</returns>
00148         public static List<Element3D> CreateSphere(Point3D center, double radius, int steps,
      IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00149         {
00150             List<Point3D> points = new List<Point3D>();
00151
00152             for (int t = 0; t <= steps; t++)
00153             {
00154                 for (int p = 0; p < steps * 2; p++)
00155                 {
00156                     double theta = Math.PI / steps * t;
00157                     double phi = Math.PI / steps * p;
00158
00159                     double x = center.X + radius * Math.Sin(theta) * Math.Cos(phi);
00160                     double y = center.Y + radius * Math.Sin(theta) * Math.Sin(phi);
00161                     double z = center.Z + radius * Math.Cos(theta);
00162
00163                     points.Add(new Point3D(x, y, z));
00164
00165                     if (t == 0 || t == steps)
00166                     {
00167                         break;
00168                     }
00169                 }
00170             }
00171
00172             List<Element3D> tbr = new List<Element3D>(4 * steps + (points.Count - 2 - 2 * steps) * 2);
00173
00174             for (int i = 0; i < points.Count - 1; i++)
00175             {
00176                 if (i == 0)
00177                 {
00178                     for (int j = 0; j < 2 * steps; j++)
00179                     {
00180                         Point3D p1 = points[i];
00181                         Point3D p3 = points[i + 1 + j];
00182                         Point3D p2 = points[i + 1 + (j + 1) % (2 * steps)];
00183
00184                         Triangle3DElement tri = new Triangle3DElement(p1, p2, p3, (center -
      p1).Normalize(), (center - p2).Normalize(), (center - p3).Normalize());
00185                         tri.Fill.AddRange(fill);
00186                         tri.Tag = tag;
00187                         tri.ZIndex = zIndex;
00188                         tbr.Add(tri);
00189                     }
00190                 }
00191                 else if (i >= points.Count - 1 - 2 * steps)
00192                 {
00193                     Point3D p1 = points[i];
00194                     Point3D p3 = points[points.Count - 1];
00195                     Point3D p2 = points[points.Count - 1 - 2 * steps + (i - (points.Count - 1 - 2 *
      steps) + 1) % (2 * steps)];
00196
00197                     Triangle3DElement tri = new Triangle3DElement(p1, p2, p3, (center -
      p1).Normalize(), (center - p2).Normalize(), (center - p3).Normalize());
00198                     tri.Fill.AddRange(fill);
00199                     tri.Tag = tag;
00200                     tri.ZIndex = zIndex;
00201                     tbr.Add(tri);
00202                 }
00203                 else
00204                 {
00205                     if ((i - 1) % (2 * steps) < 2 * steps - 1)
```

```
00206                     {
00207                         Point3D p4 = points[i + 2 * steps];
00208                         Point3D p3 = points[i + 2 * steps + 1];
00209                         Point3D p2 = points[i + 1];
00210                         Point3D p1 = points[i];
00211 
00212                         tbr.AddRange(CreateRectangle(p1, p2, p3, p4, (center - p1).Normalize(),
      (center - p2).Normalize(), (center - p3).Normalize(), (center - p4).Normalize(), fill, tag, zIndex));
00213                     }
00214                     else
00215                     {
00216                         Point3D p4 = points[i + 2 * steps];
00217                         Point3D p3 = points[(i / (2 * steps)) * 2 * steps + 1];
00218                         Point3D p2 = points[(i / (2 * steps) - 1) * 2 * steps + 1];
00219                         Point3D p1 = points[i];
00220 
00221                         tbr.AddRange(CreateRectangle(p1, p2, p3, p4, (center - p1).Normalize(),
      (center - p2).Normalize(), (center - p3).Normalize(), (center - p4).Normalize(), fill, tag, zIndex));
00222                     }
00223                 }
00224             }
00225 
00226             return tbr;
00227         }
00228 
00229 /// <summary>
00230 /// Creates a tetrahedron inscribed in a sphere.
00231 /// </summary>
00232 /// <param name="center">The centre of the tetrahedron.</param>
00233 /// <param name="radius">The radius of the sphere in which the tetrahedron is inscribed.</param>
00234 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00235 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00236 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00237 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the sphere.</returns>
00238         public static List<Element3D> CreateTetrahedron(Point3D center, double radius,
      IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00239         {
00240             Point3D tip = new Point3D(center.X, center.Y - radius, center.Z);
00241             Point3D base1 = new Point3D(Math.Sqrt(8.0 / 9) * radius + center.X, center.Y + radius / 3,
      center.Z);
00242             Point3D base2 = new Point3D(-Math.Sqrt(2.0 / 9) * radius + center.X, center.Y + radius /
      3, center.Z + Math.Sqrt(2.0 / 3) * radius);
00243             Point3D base3 = new Point3D(-Math.Sqrt(2.0 / 9) * radius + center.X, center.Y + radius /
      3, center.Z - Math.Sqrt(2.0 / 3) * radius);
00244 
00245             Triangle3DElement faceTriangle1 = new Triangle3DElement(tip, base2, base1) { Tag = tag,
      ZIndex = zIndex };
00246             faceTriangle1.Fill.AddRange(fill);
00247 
00248             Triangle3DElement faceTriangle2 = new Triangle3DElement(tip, base3, base2) { Tag = tag,
      ZIndex = zIndex };
00249             faceTriangle2.Fill.AddRange(fill);
00250 
00251             Triangle3DElement faceTriangle3 = new Triangle3DElement(tip, base1, base3) { Tag = tag,
      ZIndex = zIndex };
00252             faceTriangle3.Fill.AddRange(fill);
00253 
00254             Triangle3DElement baseTriangle = new Triangle3DElement(base1, base2, base3) { Tag = tag,
      ZIndex = zIndex };
00255             baseTriangle.Fill.AddRange(fill);
00256 
00257             return new List<Element3D>() { faceTriangle1, faceTriangle2, faceTriangle3, baseTriangle
      };
00258         }
00259 
00260 /// <summary>
00261 /// Creates a flat polygon.
00262 /// </summary>
00263 /// <param name="polygon2D">A 2D <see cref="GraphicsPath"/> representing the polygon.</param>
00264 /// <param name="triangulationResolution">The resolution that will be used to linearise curve segments
      in the <see cref="GraphicsPath"/>.</param>
00265 /// <param name="origin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
      reference system.</param>
00266 /// <param name="xAxis">A <see cref="NormalizedVector3D"/> that will correspond to the x axis of the
      2D reference system.  This will be orthonormalised to the <paramref name="yAxis"/>.</param>
00267 /// <param name="yAxis">A <see cref="NormalizedVector3D"/> that will correspond to the y axis of the
      2D reference system.</param>
00268 /// <param name="reverseTriangles">Indicates whether the order of the points (and thus the normals) of
      all the triangles returned by this method should be reversed.</param>
00269 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00270 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00271 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
```

```
returned by this method.</param>
00272 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the polygon.</returns>
00273         public static List<Element3D> CreatePolygon(GraphicsPath polygon2D, double
triangulationResolution, Point3D origin, NormalizedVector3D xAxis, NormalizedVector3D yAxis, bool
reverseTriangles, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00274         {
00275             xAxis = (xAxis - yAxis * (xAxis * yAxis)).Normalize();
00276 
00277             List<GraphicsPath> triangles = polygon2D.Triangulate(triangulationResolution,
true).ToList();
00278 
00279             List<Element3D> tbr = new List<Element3D>(triangles.Count);
00280 
00281             for (int i = 0; i < triangles.Count; i++)
00282             {
00283                 Point p1 = triangles[i].Segments[0].Point;
00284                 Point p2 = triangles[i].Segments[1].Point;
00285                 Point p3 = triangles[i].Segments[2].Point;
00286 
00287                 Point3D p13D = origin + xAxis * p1.X + yAxis * p1.Y;
00288                 Point3D p23D = origin + xAxis * p2.X + yAxis * p2.Y;
00289                 Point3D p33D = origin + xAxis * p3.X + yAxis * p3.Y;
00290 
00291                 Triangle3DElement t = !reverseTriangles ?  new Triangle3DElement(p13D, p23D, p33D) :
new Triangle3DElement(p13D, p33D, p23D);
00292                 t.Fill.AddRange(fill);
00293                 t.Tag = tag;
00294                 t.ZIndex = zIndex;
00295                 tbr.Add(t);
00296             }
00297 
00298             return tbr;
00299         }
00300 
00301 /// <summary>
00302 /// Creates a prism with the specified base.
00303 /// </summary>
00304 /// <param name="polygonBase2D">A 2D <see cref="GraphicsPath"/> representing the base of the
prism.</param>
00305 /// <param name="triangulationResolution">The resolution that will be used to linearise curve segments
in the <see cref="GraphicsPath"/>.</param>
00306 /// <param name="bottomOrigin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
reference system of the bottom base.</param>
00307 /// <param name="topOrigin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
reference system of the top base.</param>
00308 /// <param name="baseXAxis">A <see cref="NormalizedVector3D"/> that will correspond to the x axis of
the 2D reference system of the bases.  This will be orthonormalised to the <paramref
name="baseYAxis"/>.</param>
00309 /// <param name="baseYAxis">A <see cref="NormalizedVector3D"/> that will correspond to the y axis of
the 2D reference system of the bases.</param>
00310 /// <param name="fill">A collection of materials that will be applied to the <see
cref="Triangle3DElement"/>s returned by this method.</param>
00311 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
this method.</param>
00312 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
returned by this method.</param>
00313 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the prism.</returns>
00314         public static List<Element3D> CreatePrism(GraphicsPath polygonBase2D, double
triangulationResolution, Point3D bottomOrigin, Point3D topOrigin, NormalizedVector3D baseXAxis,
NormalizedVector3D baseYAxis, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00315         {
00316             baseXAxis = (baseXAxis - baseYAxis * (baseXAxis * baseYAxis)).Normalize();
00317 
00318             List<Element3D> tbr = new List<Element3D>();
00319 
00320             bool orientation = (baseXAxis ^ baseYAxis) * (bottomOrigin - topOrigin) > 0;
00321 
00322             double[,] matrix1 = Matrix3D.RotationToAlignAWithB(new NormalizedVector3D(0, 1, 0),
baseYAxis);
00323             double[,] matrix2 = Matrix3D.RotationToAlignAWithB(((Vector3D)(matrix1 * new Point3D(1, 0,
0))).Normalize(), baseXAxis);
00324 
00325             List<List<NormalizedVector3D» normals = (from el2 in
polygonBase2D.GetLinearisationPointsNormals(triangulationResolution) select (from el in el2 select
(el.X * baseXAxis + el.Y * baseYAxis).Normalize()).ToList()).ToList();
00326 
00327             polygonBase2D = polygonBase2D.Linearise(triangulationResolution);
00328 
00329             tbr.AddRange(CreatePolygon(polygonBase2D, triangulationResolution, bottomOrigin,
baseXAxis, baseYAxis, orientation, fill, tag, zIndex));
00330             tbr.AddRange(CreatePolygon(polygonBase2D, triangulationResolution, topOrigin, baseXAxis,
baseYAxis, !orientation, fill, tag, zIndex));
00331 
00332             List<List<Point3D» bottomPoints = (from el2 in polygonBase2D.GetPoints() select (from el
in el2 select (Point3D)(el.X * baseXAxis + el.Y * baseYAxis +
(Vector3D)bottomOrigin)).ToList()).ToList();
00333             List<List<Point3D» topPoints = (from el2 in polygonBase2D.GetPoints() select (from el in
```

```
    el2 select (Point3D)(el.X * baseXAxis + el.Y * baseYAxis + (Vector3D)topOrigin)).ToList()).ToList();
00334
00335             if (orientation)
00336             {
00337                 for (int i = 0; i < bottomPoints.Count; i++)
00338                 {
00339                     for (int j = 0; j < bottomPoints[i].Count - 1; j++)
00340                     {
00341                         tbr.AddRange(CreateRectangle(bottomPoints[i][j], bottomPoints[i][j + 1],
    topPoints[i][j + 1], topPoints[i][j], normals[i][j], normals[i][j + 1], normals[i][j + 1],
    normals[i][j], fill, tag, zIndex));
00342                     }
00343                 }
00344             }
00345             else
00346             {
00347                 for (int i = 0; i < bottomPoints.Count; i++)
00348                 {
00349                     for (int j = 0; j < bottomPoints[i].Count - 1; j++)
00350                     {
00351                         tbr.AddRange(CreateRectangle(bottomPoints[i][j], topPoints[i][j],
    topPoints[i][j + 1], bottomPoints[i][j + 1], normals[i][j], normals[i][j], normals[i][j + 1],
    normals[i][j + 1], fill, tag, zIndex));
00352                     }
00353                 }
00354             }
00355
00356             return tbr;
00357         }
00358
00359 /// <summary>
00360 /// Creates a wireframe from a collection of <see cref="Element3D"/>s.
00361 /// </summary>
00362 /// <param name="object3D">The collection of <see cref="Element3D"/>s.  <see cref="Line3DElement"/>s
    and <see cref="Point3DElement"/>s are ignored.</param>
00363 /// <param name="colour">The colour of the <see cref="Line3DElement"/>s returned by this
    method.</param>
00364 /// <param name="thickness">The thickness of the <see cref="Line3DElement"/>s returned by this
    method.</param>
00365 /// <param name="lineCap">The line cap of the <see cref="Line3DElement"/>s returned by this
    method.</param>
00366 /// <param name="lineDash">The line dash of the <see cref="Line3DElement"/>s returned by this
    method.</param>
00367 /// <param name="tag">A tag that will be applied to the <see cref="Line3DElement"/>s returned by this
    method.</param>
00368 /// <param name="zIndex">A z-index that will be applied to the <see cref="Line3DElement"/>s returned
    by this method.</param>
00369 /// <returns>A list of <see cref="Line3DElement"/>s that constitute the wireframe.</returns>
00370         public static List<Element3D> CreateWireframe(IEnumerable<Element3D> object3D, Colour colour,
    double thickness = 1, LineCaps lineCap = LineCaps.Butt, LineDash?  lineDash = null, string tag = null,
    int zIndex = 0)
00371         {
00372             List<Element3D> tbr = new List<Element3D>();
00373
00374             List<Point3D[]> addedLines = new List<Point3D[]>();
00375
00376             void addLine(Point3D p1, Point3D p2)
00377             {
00378                 for (int i = 0; i < addedLines.Count; i++)
00379                 {
00380                     if ((addedLines[i][0].Equals(p1, 1e-4) && addedLines[i][1].Equals(p2, 1e-4)) ||
    (addedLines[i][0].Equals(p2, 1e-4) && addedLines[i][1].Equals(p1, 1e-4)))
00381                     {
00382                         return;
00383                     }
00384                 }
00385
00386                 tbr.Add(new Line3DElement(p1, p2) { Colour = colour, Thickness = thickness, LineCap =
    lineCap, LineDash = lineDash ??  LineDash.SolidLine, Tag = tag, ZIndex = zIndex });
00387                 addedLines.Add(new Point3D[] { p1, p2 });
00388             }
00389
00390             foreach (Element3D element in object3D)
00391             {
00392                 if (element is Triangle3DElement triangle)
00393                 {
00394                     addLine(triangle.Point1, triangle.Point2);
00395                     addLine(triangle.Point2, triangle.Point3);
00396                     addLine(triangle.Point3, triangle.Point1);
00397                 }
00398             }
00399
00400             return tbr;
00401         }
00402
00403 /// <summary>
00404 /// Obtains a list of <see cref="Point3DElement"/> corresponding to the vertices of a list of <see
```

```
      cref="Element3D"/>s.
00405 /// </summary>
00406 /// <param name="object3D">The collection of <see cref="Element3D"/>s.  <see cref="Point3DElement"/>s
      are ignored.</param>
00407 /// <param name="colour">The colour of the <see cref="Point3DElement"/>s returned by this
      method.</param>
00408 /// <param name="diameter">The diameter of the <see cref="Point3DElement"/>s returned by this
      method.</param>
00409 /// <param name="tag">A tag that will be applied to the <see cref="Point3DElement"/>s returned by this
      method.</param>
00410 /// <param name="zIndex">A z-index that will be applied to the <see cref="Point3DElement"/>s returned
      by this method.</param>
00411 /// <returns>A list of <see cref="Point3DElement"/>s corresponding to the vertices of the <see
      cref="Element3D"/>s.</returns>
00412         public static List<Element3D> CreatePoints(IEnumerable<Element3D> object3D, Colour colour,
      double diameter = 1, string tag = null, int zIndex = 0)
00413         {
00414             List<Element3D> tbr = new List<Element3D>();
00415 
00416             List<Point3D> addedPoints = new List<Point3D>();
00417 
00418             void addPoint(Point3D p)
00419             {
00420                 for (int i = 0; i < addedPoints.Count; i++)
00421                 {
00422                     if (addedPoints[i].Equals(p, 1e-4))
00423                     {
00424                         return;
00425                     }
00426                 }
00427 
00428                 tbr.Add(new Point3DElement(p) { Colour = colour, Diameter = diameter, Tag = tag,
      ZIndex = zIndex });
00429                 addedPoints.Add(p);
00430             }
00431 
00432             foreach (Element3D element in object3D)
00433             {
00434                 if (!(element is Point3DElement))
00435                 {
00436                     foreach (Point3D p in element)
00437                     {
00438                         addPoint(p);
00439                     }
00440                 }
00441             }
00442 
00443             return tbr;
00444         }
00445     }
00446 }
```

8.49 Scene.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 
00021 namespace VectSharp.ThreeD
00022 {
00023 /// <summary>
00024 /// Represents a 3D scene.
00025 /// </summary>
00026     public interface IScene
00027     {
00028 /// <summary>
00029 /// The <see cref="Element3D"/>s constituting the scene.
```

```
00030 /// </summary>
00031         IEnumerable<Element3D> SceneElements { get; }
00032 
00033 /// <summary>
00034 /// Adds the specified <paramref name="element"/> to the scene.
00035 /// </summary>
00036 /// <param name="element">The <see cref="Element3D"/> to add.</param>
00037         void AddElement(Element3D element);
00038 
00039 /// <summary>
00040 /// Adds the specified <paramref name="elements"/> to the scene.
00041 /// </summary>
00042 /// <param name="elements">A collection of <see cref="Element3D"/>s to add.</param>
00043         void AddRange(IEnumerable<Element3D> elements);
00044 
00045 /// <summary>
00046 /// Replaces each element in the scene with the element returned by the <paramref
      name="replacementFunction"/>.
00047 /// </summary>
00048 /// <param name="replacementFunction">A function replacing each <see cref="Element3D"/> in the scene
      with another <see cref="Element3D"/>.</param>
00049         void Replace(Func<Element3D, Element3D> replacementFunction);
00050 
00051 /// <summary>
00052 /// Replaces each element in the scene with the element(s) returned by the <paramref
      name="replacementFunction"/>.
00053 /// </summary>
00054 /// <param name="replacementFunction">A function replacing each <see cref="Element3D"/> in the scene
      with 0 or more <see cref="Element3D"/>s.</param>
00055         void Replace(Func<Element3D, IEnumerable<Element3D>> replacementFunction);
00056 
00057 /// <summary>
00058 /// An object used to synchronise multithreaded rendering of the same scene.
00059 /// </summary>
00060         object SceneLock { get; }
00061     }
00062 
00063 /// <summary>
00064 /// Represents a 3D scene.
00065 /// </summary>
00066     public class Scene :  IScene
00067     {
00068 /// <inheritdoc/>
00069         public object SceneLock { get; }
00070 
00071         private List<Element3D> sceneElements;
00072 
00073 /// <inheritdoc/>
00074         public IEnumerable<Element3D> SceneElements => sceneElements;
00075 
00076 /// <summary>
00077 /// Creates a new <see cref="Scene"/>.
00078 /// </summary>
00079         public Scene()
00080         {
00081             sceneElements = new List<Element3D>();
00082             this.SceneLock = new object();
00083         }
00084 
00085 /// <inheritdoc/>
00086         public void AddElement(Element3D element)
00087         {
00088             this.sceneElements.Add(element);
00089         }
00090 
00091 /// <inheritdoc/>
00092         public void AddRange(IEnumerable<Element3D> elements)
00093         {
00094             this.sceneElements.AddRange(elements);
00095         }
00096 
00097 /// <inheritdoc/>
00098         public void Replace(Func<Element3D, Element3D> replacementFunction)
00099         {
00100             for (int i = 0; i < sceneElements.Count; i++)
00101             {
00102                 sceneElements[i] = replacementFunction(sceneElements[i]);
00103             }
00104         }
00105 
00106 /// <inheritdoc/>
00107         public void Replace(Func<Element3D, IEnumerable<Element3D>> replacementFunction)
00108         {
00109             List<Element3D> newElements = new List<Element3D>(sceneElements.Count);
00110 
00111             for (int i = 0; i < this.sceneElements.Count; i++)
00112             {
```

```
00113                 newElements.AddRange(replacementFunction(sceneElements[i]));
00114             }
00115 
00116             newElements.TrimExcess();
00117 
00118             this.sceneElements = newElements;
00119         }
00120     }
00121 }
```

8.50 Animation.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2022 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Collections.Immutable;
00021 using System.Linq;
00022 using VectSharp.Filters;
00023 
00024 namespace VectSharp
00025 {
00026     internal interface IAnimation
00027     {
00028         IGraphicsAction StartValue { get; }
00029         IGraphicsAction EndValue { get; }
00030         IGraphicsAction Interpolate(double position, Dictionary<string, IEasing> easings);
00031     }
00032 
00033     internal abstract class IAnimation<T> :  IAnimation where T : IGraphicsAction
00034     {
00035         public abstract T StartValue { get; }
00036         public abstract T EndValue { get; }
00037 
00038         IGraphicsAction IAnimation.StartValue => (IGraphicsAction)StartValue;
00039 
00040         IGraphicsAction IAnimation.EndValue => (IGraphicsAction)EndValue;
00041 
00042         public abstract T Interpolate(double position, Dictionary<string, IEasing> easings);
00043 
00044         IGraphicsAction IAnimation.Interpolate(double position, Dictionary<string, IEasing> easings)
00045         {
00046             return (IGraphicsAction)Interpolate(position, easings);
00047         }
00048     }
00049 
00050     internal class ConstantAnimation<T> :  IAnimation<T> where T : IGraphicsAction
00051     {
00052         public override T StartValue { get; }
00053 
00054         public override T EndValue => StartValue;
00055 
00056         public ConstantAnimation(T action)
00057         {
00058             this.StartValue = action;
00059         }
00060 
00061         public override T Interpolate(double position, Dictionary<string, IEasing> easings)
00062         {
00063             return this.StartValue;
00064         }
00065     }
00066 
00067     internal class StateAnimation :  IAnimation<StateAction>
00068     {
00069         public override StateAction StartValue { get; }
00070 
00071         public override StateAction EndValue { get; }
```

```
00072
00073         public StateAnimation(StateAction startValue, StateAction endValue)
00074         {
00075             this.StartValue = startValue;
00076             this.EndValue = endValue;
00077         }
00078
00079         public override StateAction Interpolate(double position, Dictionary<string, IEasing> easings)
00080         {
00081             if (easings != null && !string.IsNullOrEmpty(StartValue.Tag) &&
      easings.TryGetValue(StartValue.Tag, out IEasing easing))
00082             {
00083                 position = easing.Ease(position);
00084             }
00085
00086             if (position < 0.5)
00087             {
00088                 return StartValue;
00089             }
00090             else
00091             {
00092                 return EndValue;
00093             }
00094         }
00095     }
00096
00097     internal static class InterpolationUtils
00098     {
00099         public static Point InterpolatePoint(Point start, Point end, double position)
00100         {
00101             return new Point(start.X * (1 - position) + end.X * position, start.Y * (1 - position) +
      end.Y * position);
00102         }
00103
00104         public static Point InterpolatePoint(Point?  start, Point?  end, double position)
00105         {
00106             return InterpolatePoint(start ??  new Point(), end ??  new Point(), position);
00107         }
00108
00109         public static Size InterpolateSize(Size start, Size end, double position)
00110         {
00111             return new Size(start.Width * (1 - position) + end.Width * position, start.Height * (1 -
      position) + end.Height * position);
00112         }
00113
00114         public static Size InterpolateSize(Size?  start, Size?  end, double position)
00115         {
00116             return InterpolateSize(start ??  new Size(), end ??  new Size(), position);
00117         }
00118
00119         public static double InterpolateDouble(double start, double end, double position)
00120         {
00121             return start * (1 - position) + end * position;
00122         }
00123
00124         public static double InterpolateDouble(double?  start, double?  end, double position)
00125         {
00126             return start * (1 - position) + end * position ??  0;
00127         }
00128
00129         public static Colour InterpolateColour(Colour start, Colour end, double position)
00130         {
00131             return Colour.FromRgba(InterpolateDouble(start.R, end.R, position),
00132                     InterpolateDouble(start.G, end.G, position),
00133                     InterpolateDouble(start.B, end.B, position),
00134                     InterpolateDouble(start.A, end.A, position));
00135         }
00136
00137         private static HashSet<double> GetAnchors(Brush start, Brush end)
00138         {
00139             HashSet<double> anchors = new HashSet<double>();
00140
00141             if (start is GradientBrush gradientStart)
00142             {
00143                 for (int i = 0; i < gradientStart.GradientStops.Count; i++)
00144                 {
00145                     anchors.Add(gradientStart.GradientStops[i].Offset);
00146                 }
00147             }
00148
00149             if (end is GradientBrush gradientEnd)
00150             {
00151                 for (int i = 0; i < gradientEnd.GradientStops.Count; i++)
00152                 {
00153                     anchors.Add(gradientEnd.GradientStops[i].Offset);
00154                 }
00155             }
```

```
00156
00157             return anchors;
00158         }
00159
00160         private static void AddGradientStop(List<GradientStop> stops)
00161         {
00162             int minIndex = -1;
00163             double maxInterval = double.MinValue;
00164
00165             for (int i = 1; i < stops.Count; i++)
00166             {
00167                 double interval = stops[i].Offset - stops[i - 1].Offset;
00168
00169                 if (interval > maxInterval)
00170                 {
00171                     maxInterval = interval;
00172                     minIndex = i;
00173                 }
00174             }
00175
00176             if (minIndex > 0)
00177             {
00178                 stops.Insert(minIndex, new GradientStop(Colour.FromRgba((stops[minIndex - 1].Colour.R
    + stops[minIndex].Colour.R) * 0.5, (stops[minIndex - 1].Colour.G + stops[minIndex].Colour.G) * 0.5,
    (stops[minIndex - 1].Colour.B + stops[minIndex].Colour.B) * 0.5, (stops[minIndex - 1].Colour.A +
    stops[minIndex].Colour.A) * 0.5), stops[minIndex - 1].Offset + maxInterval * 0.5));
00179             }
00180         }
00181
00182         public static Brush InterpolateBrush(Brush start, Brush end, double position)
00183         {
00184             if (position <= 0)
00185             {
00186                 return start;
00187             }
00188             else if (position >= 1)
00189             {
00190                 return end;
00191             }
00192             else
00193             {
00194                 HashSet<double> anchors = GetAnchors(start, end);
00195
00196                 if (start is SolidColourBrush solidStart && end is SolidColourBrush solidEnd)
00197                 {
00198                     return new SolidColourBrush(InterpolateColour(solidStart.Colour, solidEnd.Colour,
    position));
00199                 }
00200                 else if (start is LinearGradientBrush linearStart && end is LinearGradientBrush
    linearEnd)
00201                 {
00202                     List<GradientStop> startStops = linearStart.GradientStops.ToList();
00203                     List<GradientStop> endStops = linearEnd.GradientStops.ToList();
00204
00205                     while (startStops.Count < endStops.Count)
00206                     {
00207                         AddGradientStop(startStops);
00208                     }
00209
00210                     while (startStops.Count > endStops.Count)
00211                     {
00212                         AddGradientStop(endStops);
00213                     }
00214
00215                     List<GradientStop> interpolatedStops = new List<GradientStop>();
00216
00217                     for (int i = 0; i < startStops.Count; i++)
00218                     {
00219                         interpolatedStops.Add(new GradientStop(InterpolateColour(startStops[i].Colour,
    endStops[i].Colour, position), InterpolateDouble(startStops[i].Offset, endStops[i].Offset,
    position)));
00220                     }
00221
00222                     return new LinearGradientBrush(InterpolatePoint(linearStart.StartPoint,
    linearEnd.StartPoint, position), InterpolatePoint(linearStart.EndPoint, linearEnd.EndPoint, position),
    interpolatedStops);
00223                 }
00224                 else if (start is SolidColourBrush solidStart2 && end is LinearGradientBrush
    linearEnd2)
00225                 {
00226                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00227
00228                     foreach (double anchor in anchors)
00229                     {
00230                         Colour startColour = solidStart2.Colour;
00231                         Colour endColour = linearEnd2.GradientStops.GetColourAt(anchor);
00232
```

```
00233                           gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00234                       }
00235
00236                       return new LinearGradientBrush(linearEnd2.StartPoint, linearEnd2.EndPoint,
      gradientStops);
00237                   }
00238                   else if (start is LinearGradientBrush linearStart2 && end is SolidColourBrush
      solidEnd2)
00239                   {
00240                       List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00241
00242                       foreach (double anchor in anchors)
00243                       {
00244                           Colour startColour = linearStart2.GradientStops.GetColourAt(anchor);
00245                           Colour endColour = solidEnd2.Colour;
00246
00247                           gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00248                       }
00249
00250                       return new LinearGradientBrush(linearStart2.StartPoint, linearStart2.EndPoint,
      gradientStops);
00251                   }
00252                   else if (start is RadialGradientBrush radialStart && end is RadialGradientBrush
      radialEnd)
00253                   {
00254                       List<GradientStop> startStops = radialStart.GradientStops.ToList();
00255                       List<GradientStop> endStops = radialEnd.GradientStops.ToList();
00256
00257                       while (startStops.Count < endStops.Count)
00258                       {
00259                           AddGradientStop(startStops);
00260                       }
00261
00262                       while (startStops.Count > endStops.Count)
00263                       {
00264                           AddGradientStop(endStops);
00265                       }
00266
00267                       List<GradientStop> interpolatedStops = new List<GradientStop>();
00268
00269                       for (int i = 0; i < startStops.Count; i++)
00270                       {
00271                           interpolatedStops.Add(new GradientStop(InterpolateColour(startStops[i].Colour,
      endStops[i].Colour, position), InterpolateDouble(startStops[i].Offset, endStops[i].Offset,
      position)));
00272                       }
00273
00274                       return new RadialGradientBrush(InterpolatePoint(radialStart.FocalPoint,
      radialEnd.FocalPoint, position), InterpolatePoint(radialStart.Centre, radialEnd.Centre, position),
      InterpolateDouble(radialStart.Radius, radialEnd.Radius, position), interpolatedStops);
00275                   }
00276                   else if (start is SolidColourBrush solidStart3 && end is RadialGradientBrush
      radialEnd2)
00277                   {
00278                       List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00279
00280                       foreach (double anchor in anchors)
00281                       {
00282                           Colour startColour = solidStart3.Colour;
00283                           Colour endColour = radialEnd2.GradientStops.GetColourAt(anchor);
00284
00285                           gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00286                       }
00287
00288                       return new RadialGradientBrush(radialEnd2.FocalPoint, radialEnd2.Centre,
      radialEnd2.Radius, gradientStops);
00289                   }
00290                   else if (start is RadialGradientBrush radialStart2 && end is SolidColourBrush
      solidEnd3)
00291                   {
00292                       List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00293
00294                       foreach (double anchor in anchors)
00295                       {
00296                           Colour startColour = radialStart2.GradientStops.GetColourAt(anchor);
00297                           Colour endColour = solidEnd3.Colour;
00298
00299                           gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00300                       }
00301
00302                       return new RadialGradientBrush(radialStart2.FocalPoint, radialStart2.Centre,
      radialStart2.Radius, gradientStops);
00303                   }
```

```
00304                 else if (start is LinearGradientBrush linearStart3 && end is RadialGradientBrush
      radialEnd3)
00305                 {
00306                     Point linearVector = new Point(linearStart3.EndPoint.X -
      linearStart3.StartPoint.X, linearStart3.EndPoint.Y - linearStart3.StartPoint.Y);
00307 
00308                     double modulus = linearVector.Modulus();
00309                     linearVector = new Point(linearVector.X / modulus, linearVector.Y / modulus);
00310 
00311                     double radius = radialEnd3.Radius / position;// radialEnd3.Radius * (1 + 100 * (1
      - position));
00312 
00313                     Point target = new Point(linearStart3.EndPoint.X - linearVector.X * radius,
      linearStart3.EndPoint.Y - linearVector.Y * radius);
00314 
00315                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00316 
00317                     double lOverR = (modulus + (radialEnd3.Radius - modulus) * position) /
      radialEnd3.Radius;
00318 
00319                     foreach (double anchor in anchors)
00320                     {
00321                         double offset = (1 - lOverR * position) + anchor * position * lOverR;
00322 
00323                         Colour startColour = linearStart3.GradientStops.GetColourAt(anchor);
00324                         Colour endColour = radialEnd3.GradientStops.GetColourAt(anchor);
00325                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), offset));
00326                     }
00327 
00328                     return new RadialGradientBrush(InterpolatePoint(target, radialEnd3.FocalPoint,
      position), InterpolatePoint(target, radialEnd3.Centre, position), InterpolateDouble(radius,
      radialEnd3.Radius, position), gradientStops);
00329                 }
00330                 else if (start is RadialGradientBrush && end is LinearGradientBrush)
00331                 {
00332                     return InterpolateBrush(end, start, 1 - position);
00333                 }
00334                 else
00335                 {
00336                     if (position < 0.5)
00337                     {
00338                         return start;
00339                     }
00340                     else
00341                     {
00342                         return end;
00343                     }
00344                 }
00345             }
00346         }
00347 
00348         public static int InterpolateInt(int start, int end, double position)
00349         {
00350             return (int)Math.Round(start * (1 - position) + end * position);
00351         }
00352 
00353         public static Font InterpolateFont(Font start, Font end, double position)
00354         {
00355             Font tbr = new Font(position < 0.5 ?  start.FontFamily :  end.FontFamily,
      InterpolateDouble(start.FontSize, end.FontSize, position), start.Underline != null || end.Underline !=
      null);
00356 
00357             if (start.Underline != null && end.Underline == null)
00358             {
00359                 tbr.Underline.FollowItalicAngle = start.Underline.FollowItalicAngle;
00360                 tbr.Underline.LineCap = start.Underline.LineCap;
00361                 tbr.Underline.Position = start.Underline.Position;
00362                 tbr.Underline.SkipDescenders = start.Underline.SkipDescenders;
00363                 tbr.Underline.Thickness = InterpolateDouble(start.Underline.Thickness, 0, position);
00364             }
00365             else if (start.Underline == null && end.Underline != null)
00366             {
00367                 tbr.Underline.FollowItalicAngle = end.Underline.FollowItalicAngle;
00368                 tbr.Underline.LineCap = end.Underline.LineCap;
00369                 tbr.Underline.Position = end.Underline.Position;
00370                 tbr.Underline.SkipDescenders = end.Underline.SkipDescenders;
00371                 tbr.Underline.Thickness = InterpolateDouble(0, end.Underline.Thickness, position);
00372             }
00373             else if (start.Underline != null && end.Underline != null)
00374             {
00375                 tbr.Underline.FollowItalicAngle = position < 0.5 ?  start.Underline.FollowItalicAngle
      :  end.Underline.FollowItalicAngle;
00376                 tbr.Underline.LineCap = position < 0.5 ?  start.Underline.LineCap :
      end.Underline.LineCap;
00377                 tbr.Underline.Position = InterpolateDouble(start.Underline.Position,
      end.Underline.Position, position);
```

```
00378                 tbr.Underline.SkipDescenders = position < 0.5 ?  start.Underline.SkipDescenders :
      end.Underline.SkipDescenders;
00379                 tbr.Underline.Thickness = InterpolateDouble(start.Underline.Thickness, 0, position);
00380             }
00381
00382             return tbr;
00383         }
00384     }
00385
00386     internal class TransformAnimation :  IAnimation<TransformAction>
00387     {
00388         public override TransformAction StartValue { get; }
00389
00390         public override TransformAction EndValue { get; }
00391
00392         public TransformAnimation(TransformAction startValue, TransformAction endValue)
00393         {
00394             this.StartValue = startValue;
00395             this.EndValue = endValue;
00396         }
00397         public override TransformAction Interpolate(double position, Dictionary<string, IEasing>
      easings)
00398         {
00399             if (easings != null && !string.IsNullOrEmpty(StartValue.Tag) &&
      easings.TryGetValue(StartValue.Tag, out IEasing easing))
00400             {
00401                 position = easing.Ease(position);
00402             }
00403
00404             if (position <= 0)
00405             {
00406                 return this.StartValue;
00407             }
00408             else if (position >= 1)
00409             {
00410                 return this.EndValue;
00411             }
00412             else
00413             {
00414                 if (this.StartValue.Delta != null && this.EndValue.Delta != null)
00415                 {
00416                     return new
      TransformAction(InterpolationUtils.InterpolatePoint(this.StartValue.Delta, this.EndValue.Delta,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00417                 }
00418                 else if (this.StartValue.Scale != null && this.EndValue.Scale != null)
00419                 {
00420                     return new
      TransformAction(InterpolationUtils.InterpolateSize(this.StartValue.Scale, this.EndValue.Scale,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00421                 }
00422                 else if (this.StartValue.Angle != null && this.EndValue.Angle != null)
00423                 {
00424                     return new
      TransformAction(InterpolationUtils.InterpolateDouble(this.StartValue.Angle, this.EndValue.Angle,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00425                 }
00426                 else
00427                 {
00428                     double[,] startMatrix = this.StartValue.GetMatrix();
00429                     double[,] endMatrix = this.EndValue.GetMatrix();
00430
00431                     double[,] tbrMatrix = new double[startMatrix.GetLength(0),
      startMatrix.GetLength(1)];
00432
00433                     for (int i = 0; i < startMatrix.GetLength(0); i++)
00434                     {
00435                         for (int j = 0; j < startMatrix.GetLength(1); j++)
00436                         {
00437                             tbrMatrix[i, j] = startMatrix[i, j] * (1 - position) + endMatrix[i, j] *
      position;
00438                         }
00439                     }
00440
00441                     return new TransformAction(tbrMatrix, position < 0.5 ?  this.StartValue.Tag :
      this.EndValue.Tag);
00442                 }
00443             }
00444         }
00445     }
00446
00447     internal abstract class PrintableActionAnimation<T> :  IAnimation<T> where T : IPrintableAction,
      IGraphicsAction
00448     {
00449         public override T StartValue { get; }
00450         public override T EndValue { get; }
00451
```

```
00452         protected abstract T InterpolateConcrete((double, LineJoins, LineCaps, LineDash, Brush, Brush)
      commonElements, double position, Dictionary<string, IEasing> easings);
00453
00454         public PrintableActionAnimation(T startValue, T endValue)
00455         {
00456             this.StartValue = startValue;
00457             this.EndValue = endValue;
00458         }
00459
00460         public override T Interpolate(double position, Dictionary<string, IEasing> easings)
00461         {
00462             string tag = (StartValue as IPrintableAction)?.Tag ??  (StartValue as
      IGraphicsAction)?.Tag;
00463
00464             if (easings != null && !string.IsNullOrEmpty(tag) && easings.TryGetValue(tag, out IEasing
      easing))
00465             {
00466                 position = easing.Ease(position);
00467             }
00468
00469             if (position <= 0)
00470             {
00471                 return StartValue;
00472             }
00473             else if (position >= 1)
00474             {
00475                 return EndValue;
00476             }
00477             else
00478             {
00479                 T tbr = InterpolateConcrete(InterpolateCommon(position), position, easings);
00480                 ((IGraphicsAction)tbr).Tag = ((IGraphicsAction)this.StartValue).Tag;
00481                 return tbr;
00482             }
00483         }
00484
00485         protected (double, LineJoins, LineCaps, LineDash, Brush, Brush) InterpolateCommon(double
      position)
00486         {
00487             double lineWidth = InterpolationUtils.InterpolateDouble(this.StartValue.LineWidth,
      this.EndValue.LineWidth, position);
00488             LineJoins lineJoin = position < 0.5 ?  this.StartValue.LineJoin :  this.EndValue.LineJoin;
00489             LineCaps lineCap = position < 0.5 ?  this.StartValue.LineCap :  this.EndValue.LineCap;
00490             LineDash lineDash = new
      LineDash(InterpolationUtils.InterpolateDouble(this.StartValue.LineDash.UnitsOn,
      this.EndValue.LineDash.UnitsOn, position),
00491                 InterpolationUtils.InterpolateDouble(this.StartValue.LineDash.UnitsOff,
      this.EndValue.LineDash.UnitsOff, position),
00492                 InterpolationUtils.InterpolateDouble(this.StartValue.LineDash.Phase,
      this.EndValue.LineDash.Phase, position));
00493
00494             Brush stroke = null;
00495
00496             if (this.StartValue.Stroke != null && this.EndValue.Stroke != null)
00497             {
00498                 stroke = InterpolationUtils.InterpolateBrush(this.StartValue.Stroke,
      this.EndValue.Stroke, position);
00499             }
00500
00501             Brush fill = null;
00502
00503             if (this.StartValue.Fill != null && this.EndValue.Fill != null)
00504             {
00505                 fill = InterpolationUtils.InterpolateBrush(this.StartValue.Fill, this.EndValue.Fill,
      position);
00506             }
00507
00508             return (lineWidth, lineJoin, lineCap, lineDash, stroke, fill);
00509         }
00510     }
00511
00512     internal class RectangleActionAnimation :  PrintableActionAnimation<RectangleAction>
00513     {
00514         public RectangleActionAnimation(RectangleAction startValue, RectangleAction endValue) :
      base(startValue, endValue) { }
00515
00516         protected override RectangleAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00517         {
00518             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
      Brush fill) = commonElements;
00519
00520             return new RectangleAction(InterpolationUtils.InterpolatePoint(this.StartValue.TopLeft,
      this.EndValue.TopLeft, position),
00521                         InterpolationUtils.InterpolateSize(this.StartValue.Size, this.EndValue.Size,
      position),
00522                         fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position < 0.5 ?
```

```
      this.StartValue.Tag :  this.EndValue.Tag);
00523         }
00524     }
00525
00526     internal class RasterImageActionAnimation :  PrintableActionAnimation<RasterImageAction>
00527     {
00528         public RasterImageActionAnimation(RasterImageAction startValue, RasterImageAction endValue) :
      base(startValue, endValue) { }
00529
00530         protected override RasterImageAction InterpolateConcrete((double, LineJoins, LineCaps,
      LineDash, Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00531         {
00532             return new RasterImageAction(InterpolationUtils.InterpolateInt(this.StartValue.SourceX,
      this.EndValue.SourceX, position),
00533                 InterpolationUtils.InterpolateInt(this.StartValue.SourceY, this.EndValue.SourceY,
      position),
00534                 InterpolationUtils.InterpolateInt(this.StartValue.SourceWidth,
      this.EndValue.SourceWidth, position),
00535                 InterpolationUtils.InterpolateInt(this.StartValue.SourceHeight,
      this.EndValue.SourceHeight, position),
00536                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationX,
      this.EndValue.DestinationX, position),
00537                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationY,
      this.EndValue.DestinationY, position),
00538                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationWidth,
      this.EndValue.DestinationWidth, position),
00539                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationHeight,
      this.EndValue.DestinationHeight, position), position < 0.5 ?  this.StartValue.Image :
      this.EndValue.Image
00540                 , position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00541         }
00542     }
00543
00544     internal class TextActionAnimation :  PrintableActionAnimation<TextAction>
00545     {
00546         public TextActionAnimation(TextAction startValue, TextAction endValue) :  base(startValue,
      endValue) { }
00547
00548         protected override TextAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00549         {
00550             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
      Brush fill) = commonElements;
00551
00552             return new TextAction(InterpolationUtils.InterpolatePoint(this.StartValue.Origin,
      this.EndValue.Origin, position),
00553                 position < 0.5 ?  this.StartValue.Text :  this.EndValue.Text,
00554                 InterpolationUtils.InterpolateFont(this.StartValue.Font, this.EndValue.Font,
      position),
00555                 position < 0.5 ?  this.StartValue.TextBaseline :  this.EndValue.TextBaseline,
00556                 fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position < 0.5 ?
      this.StartValue.Tag :  this.EndValue.Tag);
00557         }
00558     }
00559
00560     internal class PathActionAnimation :  PrintableActionAnimation<PathAction>
00561     {
00562         public override PathAction StartValue { get; }
00563         public override PathAction EndValue { get; }
00564
00565         private bool AreTheyTheSame(GraphicsPath pth1, GraphicsPath pth2)
00566         {
00567             if (pth1.Segments.Count != pth2.Segments.Count)
00568             {
00569                 return false;
00570             }
00571
00572             for (int i = 0; i < pth1.Segments.Count; i++)
00573             {
00574                 if (pth1.Segments[i].Type != pth2.Segments[i].Type)
00575                 {
00576                     return false;
00577                 }
00578             }
00579
00580             return true;
00581         }
00582
00583         private Point GetCenter(GraphicsPath path)
00584         {
00585             List<Point> points = path.GetPoints().ToList().Aggregate(new List<Point>(), (a, b) => {
      a.AddRange(b); return a; });
00586
00587             double cX = 0;
00588             double cY = 0;
00589
00590             for (int i = 0; i < points.Count; i++)
```

```
00591             {
00592                 cX += points[i].X;
00593                 cY += points[i].Y;
00594             }
00595 
00596             return new Point(cX / points.Count, cY / points.Count);
00597         }
00598 
00599         private double ComparePaths(GraphicsPath path1, GraphicsPath path2)
00600         {
00601             Point c1 = GetCenter(path1);
00602             Point c2 = GetCenter(path2);
00603 
00604             return (c1.X - c2.X) * (c1.X - c2.X) + (c1.Y - c2.Y) * (c1.Y - c2.Y);
00605         }
00606 
00607         private List<(int, int)> GetBestAssignment(List<GraphicsPath> figures1, List<GraphicsPath>
      figures2)
00608         {
00609             List<(int, int)> bestAssignment = new List<(int, int)>();
00610 
00611             double[,] distMat = new double[figures1.Count, figures2.Count];
00612 
00613             for (int i = 0; i < figures1.Count; i++)
00614             {
00615                 for (int j = 0; j < figures2.Count; j++)
00616                 {
00617                     distMat[i, j] = ComparePaths(figures1[i], figures2[j]);
00618                 }
00619             }
00620 
00621             List<int> missingIndices1 = Enumerable.Range(0, figures1.Count).ToList();
00622             List<int> missingIndices2 = Enumerable.Range(0, figures2.Count).ToList();
00623 
00624             if (figures1.Count > figures2.Count)
00625             {
00626                 while (missingIndices2.Count > 0)
00627                 {
00628                     double minValue = double.MaxValue;
00629                     (int, int) bestPair = (-1, -1);
00630 
00631                     for (int i = 0; i < missingIndices1.Count; i++)
00632                     {
00633                         for (int j = 0; j < missingIndices2.Count; j++)
00634                         {
00635                             if (distMat[missingIndices1[i], missingIndices2[j]] < minValue)
00636                             {
00637                                 minValue = distMat[missingIndices1[i], missingIndices2[j]];
00638                                 bestPair = (missingIndices1[i], missingIndices2[j]);
00639                             }
00640                         }
00641                     }
00642 
00643                     bestAssignment.Add(bestPair);
00644                     missingIndices1.Remove(bestPair.Item1);
00645                     missingIndices2.Remove(bestPair.Item2);
00646                 }
00647             }
00648             else
00649             {
00650                 while (missingIndices1.Count > 0)
00651                 {
00652                     double minValue = double.MaxValue;
00653                     (int, int) bestPair = (-1, -1);
00654 
00655                     for (int i = 0; i < missingIndices1.Count; i++)
00656                     {
00657                         for (int j = 0; j < missingIndices2.Count; j++)
00658                         {
00659                             if (distMat[missingIndices1[i], missingIndices2[j]] < minValue)
00660                             {
00661                                 minValue = distMat[missingIndices1[i], missingIndices2[j]];
00662                                 bestPair = (missingIndices1[i], missingIndices2[j]);
00663                             }
00664                         }
00665                     }
00666 
00667                     bestAssignment.Add(bestPair);
00668                     missingIndices1.Remove(bestPair.Item1);
00669                     missingIndices2.Remove(bestPair.Item2);
00670                 }
00671             }
00672 
00673             return bestAssignment;
00674         }
00675 
00676         private void IncreasePoints(List<Point> points, bool isClosed)
```

```
00677         {
00678             if (points.Count > 1)
00679             {
00680                 double maxLength = double.MinValue;
00681                 int maxIndex = -1;
00682 
00683                 double target = isClosed ?  points.Count :  points.Count - 1;
00684 
00685                 for (int i = 0; i < target; i++)
00686                 {
00687                     double length = (points[i].X - points[(i + 1) % points.Count].X) * (points[i].X -
      points[(i + 1) % points.Count].X) + (points[i].Y - points[(i + 1) % points.Count].Y) * (points[i].Y -
      points[(i + 1) % points.Count].Y);
00688 
00689                     if (length > maxLength)
00690                     {
00691                         maxLength = length;
00692                         maxIndex = i;
00693                     }
00694                 }
00695 
00696                 points.Insert(maxIndex + 1, new Point((points[maxIndex].X + points[(maxIndex + 1) %
      points.Count].X) * 0.5, (points[maxIndex].Y + points[(maxIndex + 1) % points.Count].Y) * 0.5));
00697             }
00698             else
00699             {
00700                 points.Add(points[0]);
00701             }
00702 
00703         }
00704 
00705         public PathActionAnimation(PathAction startValue, PathAction endValue, double
      linearisationResolution) :  base(startValue, endValue)
00706         {
00707             GraphicsPath startPath = startValue.Path.ConvertArcsToBeziers();
00708             GraphicsPath endPath = endValue.Path.ConvertArcsToBeziers();
00709 
00710             if (AreTheyTheSame(startPath, endPath))
00711             {
00712                 this.StartValue = new PathAction(startPath, startValue.Fill, startValue.Stroke,
      startValue.LineWidth, startValue.LineCap, startValue.LineJoin, startValue.LineDash, startValue.Tag,
      startValue.FillRule, startValue.IsClipping);
00713                 this.EndValue = new PathAction(endPath, endValue.Fill, endValue.Stroke,
      endValue.LineWidth, endValue.LineCap, endValue.LineJoin, endValue.LineDash, endValue.Tag,
      endValue.FillRule, endValue.IsClipping);
00714             }
00715             else
00716             {
00717                 List<GraphicsPath> startFigures = startPath.GetFigures().ToList();
00718                 List<GraphicsPath> endFigures = endPath.GetFigures().ToList();
00719 
00720                 List<(int, int)> correspondences = GetBestAssignment(startFigures, endFigures);
00721 
00722                 HashSet<int> startFiguresIndices = new HashSet<int>(Enumerable.Range(0,
      startFigures.Count));
00723                 HashSet<int> endFiguresIndices = new HashSet<int>(Enumerable.Range(0,
      endFigures.Count));
00724 
00725                 for (int i = 0; i < correspondences.Count; i++)
00726                 {
00727                     startFiguresIndices.Remove(correspondences[i].Item1);
00728                     endFiguresIndices.Remove(correspondences[i].Item2);
00729                 }
00730 
00731                 foreach (int i in startFiguresIndices)
00732                 {
00733                     Point center = GetCenter(startFigures[i]);
00734 
00735                     GraphicsPath path = new GraphicsPath().MoveTo(center).LineTo(center);
00736                     if (startFigures[i].Segments[startFigures[i].Segments.Count - 1].Type ==
      SegmentType.Close)
00737                     {
00738                         path.Close();
00739                     }
00740 
00741                     endFigures.Add(path);
00742                     correspondences.Add((i, endFigures.Count - 1));
00743                 }
00744 
00745 
00746                 foreach (int i in endFiguresIndices)
00747                 {
00748                     Point center = GetCenter(endFigures[i]);
00749 
00750                     GraphicsPath path = new GraphicsPath().MoveTo(center).LineTo(center);
00751                     if (endFigures[i].Segments[endFigures[i].Segments.Count - 1].Type ==
      SegmentType.Close)
```

```
00752                     {
00753                         path.Close();
00754                     }
00755 
00756                     startFigures.Add(path);
00757                     correspondences.Add((startFigures.Count - 1, i));
00758                 }
00759 
00760 
00761                 startPath = new GraphicsPath();
00762                 endPath = new GraphicsPath();
00763 
00764                 for (int i = 0; i < startFigures.Count; i++)
00765                 {
00766                     double startLength = startFigures[correspondences[i].Item1].MeasureLength();
00767                     double endLength = endFigures[correspondences[i].Item2].MeasureLength();
00768 
00769                     double maxLength = Math.Max(startLength, endLength);
00770                     int numPoints = (int)Math.Ceiling(maxLength / linearisationResolution);
00771 
00772                     double startResolution = startLength / maxLength * linearisationResolution;
00773                     double endResolution = endLength / maxLength * linearisationResolution;
00774 
00775                     if (startResolution <= 0)
00776                     {
00777                         startResolution = 1;
00778                     }
00779 
00780                     if (endResolution <= 0)
00781                     {
00782                         endResolution = 1;
00783                     }
00784 
00785                     List<Point> startFigure =
      startFigures[correspondences[i].Item1].Linearise(startResolution).GetPoints().First();
00786                     bool startIsClosed =
      startFigures[correspondences[i].Item1].Segments[startFigures[correspondences[i].Item1].Segments.Count
      - 1].Type == SegmentType.Close;
00787 
00788                     List<Point> endFigure =
      endFigures[correspondences[i].Item2].Linearise(endResolution).GetPoints().First();
00789                     bool endIsClosed =
      endFigures[correspondences[i].Item2].Segments[endFigures[correspondences[i].Item2].Segments.Count -
      1].Type == SegmentType.Close;
00790 
00791                     while (startFigure.Count < numPoints)
00792                     {
00793                         IncreasePoints(startFigure, startIsClosed);
00794                     }
00795 
00796                     while (endFigure.Count < numPoints)
00797                     {
00798                         IncreasePoints(endFigure, endIsClosed);
00799                     }
00800 
00801 
00802                     double minVariance = double.MaxValue;
00803                     int bestShift = -1;
00804 
00805                     for (int j = 0; j < numPoints; j++)
00806                     {
00807                         double average = 0;
00808                         //double averageSq = 0;
00809 
00810                         for (int k = 0; k < numPoints; k++)
00811                         {
00812                             double dist = (startFigure[k].X - endFigure[(k + j) % numPoints].X) *
      (startFigure[k].X - endFigure[(k + j) % numPoints].X) + (startFigure[k].Y - endFigure[(k + j) %
      numPoints].Y) * (startFigure[k].Y - endFigure[(k + j) % numPoints].Y);
00813                             average += dist;
00814                             //averageSq += dist * dist;
00815                         }
00816 
00817                         //double variance = averageSq / numPoints - (average / numPoints) * (average /
      numPoints);
00818 
00819                         if (average < minVariance)
00820                         {
00821                             minVariance = average;
00822                             bestShift = j;
00823                         }
00824                     }
00825 
00826                     startPath.MoveTo(startFigure[0]);
00827                     endPath.MoveTo(endFigure[bestShift % numPoints]);
00828 
00829                     for (int j = 0; j < numPoints; j++)
```

```
00830                     {
00831                         startPath.LineTo(startFigure[j]);
00832                         endPath.LineTo(endFigure[(j + bestShift) % numPoints]);
00833                     }
00834 
00835                     if (startIsClosed && endIsClosed)
00836                     {
00837                         startPath.Close();
00838                         endPath.Close();
00839                     }
00840                     else if (startIsClosed && !endIsClosed)
00841                     {
00842                         startPath.LineTo(startFigure[0]);
00843                         endPath.LineTo(endFigure[endFigure.Count - 1]);
00844                     }
00845                     else if (!startIsClosed && endIsClosed)
00846                     {
00847                         startPath.LineTo(startFigure[startFigure.Count - 1]);
00848                         endPath.LineTo(endFigure[0]);
00849                     }
00850                 }
00851 
00852                 this.StartValue = new PathAction(startPath, startValue.Fill, startValue.Stroke,
      startValue.LineWidth, startValue.LineCap, startValue.LineJoin, startValue.LineDash, startValue.Tag,
      startValue.FillRule, startValue.IsClipping);
00853                 this.EndValue = new PathAction(endPath, endValue.Fill, endValue.Stroke,
      endValue.LineWidth, endValue.LineCap, endValue.LineJoin, endValue.LineDash, endValue.Tag,
      endValue.FillRule, endValue.IsClipping);
00854 
00855             }
00856         }
00857 
00858         private static Segment InterpolateSegment(Segment start, Segment end, double position)
00859         {
00860             if (start.Type == end.Type)
00861             {
00862                 if (start.Type == SegmentType.Arc)
00863                 {
00864                     ArcSegment startA = start as ArcSegment;
00865                     ArcSegment endA = end as ArcSegment;
00866 
00867                     return new ArcSegment(InterpolationUtils.InterpolatePoint(startA.Points[0],
      endA.Points[0], position), InterpolationUtils.InterpolateDouble(startA.Radius, endA.Radius, position),
      InterpolationUtils.InterpolateDouble(startA.StartAngle, endA.StartAngle, position),
      InterpolationUtils.InterpolateDouble(startA.EndAngle, endA.EndAngle, position));
00868                 }
00869                 else if (start.Type == SegmentType.Move)
00870                 {
00871                     return new MoveSegment(InterpolationUtils.InterpolatePoint(start.Point, end.Point,
      position));
00872                 }
00873                 else if (start.Type == SegmentType.Line)
00874                 {
00875                     return new LineSegment(InterpolationUtils.InterpolatePoint(start.Point, end.Point,
      position));
00876                 }
00877                 else if (start.Type == SegmentType.Close)
00878                 {
00879                     return new CloseSegment();
00880                 }
00881                 else if (start.Type == SegmentType.CubicBezier)
00882                 {
00883                     CubicBezierSegment startC = start as CubicBezierSegment;
00884                     CubicBezierSegment endC = end as CubicBezierSegment;
00885 
00886                     return new
      CubicBezierSegment(InterpolationUtils.InterpolatePoint(startC.Points[0], endC.Points[0], position),
00887                         InterpolationUtils.InterpolatePoint(startC.Points[1], endC.Points[1],
      position),
00888                         InterpolationUtils.InterpolatePoint(startC.Points[2], endC.Points[2],
      position));
00889                 }
00890                 else
00891                 {
00892                     throw new NotImplementedException();
00893                 }
00894             }
00895             else
00896             {
00897                 return position < 0.5 ?  start :  end;
00898             }
00899         }
00900 
00901         protected override PathAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00902         {
00903             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
```

```
    Brush fill) = commonElements;
00904
00905             GraphicsPath path = new GraphicsPath();
00906
00907             for (int i = 0; i < this.StartValue.Path.Segments.Count; i++)
00908             {
00909                 path.Segments.Add(InterpolateSegment(this.StartValue.Path.Segments[i],
    this.EndValue.Path.Segments[i], position));
00910             }
00911
00912             return new PathAction(path, fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position
    < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag, position < 0.5 ?  this.StartValue.FillRule :
    this.EndValue.FillRule, position < 0.5 ?  this.StartValue.IsClipping :  this.EndValue.IsClipping);
00913         }
00914     }
00915
00916     internal class FilteredGraphicsAnimation :  PrintableActionAnimation<FilteredGraphicsAction>
00917     {
00918         private IGraphicsAnimation GraphicsAnimation { get; }
00919         private IGraphicsAnimation MaskAnimation { get; }
00920
00921         public FilteredGraphicsAnimation(FilteredGraphicsAction startValue, FilteredGraphicsAction
    endValue, double linearisationResolution) :  base(startValue, endValue)
00922         {
00923             if (startValue.Content == endValue.Content)
00924             {
00925                 this.GraphicsAnimation = new ConstantGraphicsAnimation(startValue.Content);
00926             }
00927             else
00928             {
00929                 this.GraphicsAnimation = new GraphicsAnimation(startValue.Content, endValue.Content,
    linearisationResolution);
00930             }
00931
00932             if (startValue.Filter is MaskFilter startM && endValue.Filter is MaskFilter endM)
00933             {
00934                 if (startM.Mask == endM.Mask)
00935                 {
00936                     this.MaskAnimation = new ConstantGraphicsAnimation(startM.Mask);
00937                 }
00938                 else
00939                 {
00940                     this.MaskAnimation = new GraphicsAnimation(startM.Mask, endM.Mask,
    linearisationResolution);
00941                 }
00942             }
00943         }
00944
00945         protected override FilteredGraphicsAction InterpolateConcrete((double, LineJoins, LineCaps,
    LineDash, Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00946         {
00947             Graphics gpr = this.GraphicsAnimation.Interpolate(position, easings);
00948
00949             if (this.StartValue.Filter.GetType() == this.EndValue.Filter.GetType())
00950             {
00951                 if (this.StartValue.Filter is BoxBlurFilter startBBF && this.EndValue.Filter is
    BoxBlurFilter endBBF)
00952                 {
00953                     BoxBlurFilter filter = new
    BoxBlurFilter(InterpolationUtils.InterpolateDouble(startBBF.BoxRadius, endBBF.BoxRadius, position));
00954
00955                     return new FilteredGraphicsAction(gpr, filter);
00956                 }
00957                 else if (this.StartValue.Filter is ColourMatrixFilter startCMF && this.EndValue.Filter
    is ColourMatrixFilter endCMF)
00958                 {
00959                     double[,] colourMatrix = new double[5, 5];
00960
00961                     for (int i = 0; i < 5; i++)
00962                     {
00963                         for (int j = 0; j < 5; j++)
00964                         {
00965                             colourMatrix[i, j] =
    InterpolationUtils.InterpolateDouble(startCMF.ColourMatrix[i, j], endCMF.ColourMatrix[i, j],
    position);
00966                         }
00967                     }
00968
00969                     ColourMatrixFilter filter = new ColourMatrixFilter(new
    ColourMatrix(colourMatrix));
00970
00971                     return new FilteredGraphicsAction(gpr, filter);
00972                 }
00973                 else if (this.StartValue.Filter is ConvolutionFilter startConvo &&
    this.EndValue.Filter is ConvolutionFilter endConvo)
00974                 {
00975                     if (startConvo.Kernel.GetLength(0) == endConvo.Kernel.GetLength(0) &&
```

```
    startConvo.Kernel.GetLength(1) == endConvo.Kernel.GetLength(1))
00976                     {
00977                         double[,] kernel = new double[startConvo.Kernel.GetLength(0),
    startConvo.Kernel.GetLength(1)];
00978 
00979                         for (int i = 0; i < kernel.GetLength(0); i++)
00980                         {
00981                             for (int j = 0; j < kernel.GetLength(1); j++)
00982                             {
00983                                 kernel[i, j] =
    InterpolationUtils.InterpolateDouble(startConvo.Kernel[i, j], endConvo.Kernel[i, j], position);
00984                             }
00985                         }
00986 
00987                         ConvolutionFilter filter = new ConvolutionFilter(kernel,
    InterpolationUtils.InterpolateDouble(startConvo.Scale, endConvo.Scale, position), position < 0.5 ?
    startConvo.PreserveAlpha :  endConvo.PreserveAlpha,
    InterpolationUtils.InterpolateDouble(startConvo.Normalisation, endConvo.Normalisation, position),
    InterpolationUtils.InterpolateDouble(startConvo.Bias, endConvo.Bias, position));
00988                         return new FilteredGraphicsAction(gpr, filter);
00989                     }
00990                     else
00991                     {
00992                         return position < 0.5 ?  this.StartValue :  this.EndValue;
00993                     }
00994                 }
00995                 else if (this.StartValue.Filter is GaussianBlurFilter startGauss &&
    this.EndValue.Filter is GaussianBlurFilter endGauss)
00996                 {
00997                     GaussianBlurFilter filter = new
    GaussianBlurFilter(InterpolationUtils.InterpolateDouble(startGauss.StandardDeviation,
    endGauss.StandardDeviation, position));
00998                     return new FilteredGraphicsAction(gpr, filter);
00999                 }
01000                 else if (this.StartValue.Filter is MaskFilter startMask && this.EndValue.Filter is
    MaskFilter endMask)
01001                 {
01002                     Graphics mask = MaskAnimation.Interpolate(position, easings);
01003                     MaskFilter filter = new MaskFilter(mask);
01004                     return new FilteredGraphicsAction(gpr, filter);
01005                 }
01006                 else
01007                 {
01008                     return position < 0.5 ?  this.StartValue :  this.EndValue;
01009                 }
01010             }
01011             else
01012             {
01013                 return position < 0.5 ?  this.StartValue :  this.EndValue;
01014             }
01015         }
01016     }
01017 
01018     internal interface IGraphicsAnimation
01019     {
01020         Graphics StartValue { get; }
01021         Graphics EndValue { get; }
01022         Graphics Interpolate(double position, Dictionary<string, IEasing> easings);
01023     }
01024 
01025     internal class ConstantGraphicsAnimation :  IGraphicsAnimation
01026     {
01027         public Graphics StartValue { get; }
01028         public Graphics EndValue => StartValue;
01029         public Graphics Interpolate(double position, Dictionary<string, IEasing> easings) =>
    this.StartValue;
01030 
01031         public ConstantGraphicsAnimation(Graphics startValue)
01032         {
01033             this.StartValue = startValue;
01034         }
01035     }
01036 
01037     internal class GraphicsAnimation :  IGraphicsAnimation
01038     {
01039         public Graphics StartValue { get; }
01040         public Graphics EndValue { get; }
01041 
01042         private List<IAnimation> StartAnimations { get; }
01043         private List<IAnimation> EndAnimations { get; }
01044 
01045         public GraphicsAnimation(Graphics startValue, Graphics endValue, double
    linearisationResolution)
01046         {
01047             this.StartValue = startValue;
01048             this.EndValue = endValue;
01049 
```

```
01050             Dictionary<string, IGraphicsAction> startTaggedActions = new Dictionary<string,
      IGraphicsAction>();
01051
01052             foreach (IGraphicsAction action in startValue.Actions)
01053             {
01054                 if (!string.IsNullOrEmpty(action.Tag))
01055                 {
01056                     startTaggedActions[action.Tag] = action;
01057                 }
01058             }
01059
01060             Dictionary<string, IGraphicsAction> endTaggedActions = new Dictionary<string,
      IGraphicsAction>();
01061
01062             foreach (IGraphicsAction action in endValue.Actions)
01063             {
01064                 if (!string.IsNullOrEmpty(action.Tag))
01065                 {
01066                     endTaggedActions[action.Tag] = action;
01067                 }
01068             }
01069
01070             List<IAnimation> startAnimations = new List<IAnimation>();
01071
01072             Dictionary<string, IAnimation> computedAnimations = new Dictionary<string, IAnimation>();
01073
01074             for (int i = 0; i < this.StartValue.Actions.Count; i++)
01075             {
01076                 IGraphicsAction startAction = this.StartValue.Actions[i];
01077
01078                 if (!string.IsNullOrEmpty(this.StartValue.Actions[i].Tag) &&
      endTaggedActions.TryGetValue(this.StartValue.Actions[i].Tag, out IGraphicsAction endAction) &&
      startAction.GetType() == endAction.GetType())
01079                 {
01080                     if (startAction is TransformAction startT && endAction is TransformAction endT)
01081                     {
01082                         IAnimation animation = new TransformAnimation(startT, endT);
01083
01084                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01085
01086                         startAnimations.Add(animation);
01087                     }
01088                     else if (startAction is StateAction startS && endAction is StateAction endS)
01089                     {
01090                         IAnimation animation = new StateAnimation(startS, endS);
01091                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01092                         startAnimations.Add(animation);
01093                     }
01094                     else if (startAction is TextAction startX && endAction is TextAction endX)
01095                     {
01096                         IAnimation animation = new TextActionAnimation(startX, endX);
01097                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01098                         startAnimations.Add(animation);
01099                     }
01100                     else if (startAction is RectangleAction startR && endAction is RectangleAction
      endR)
01101                     {
01102                         IAnimation animation = new RectangleActionAnimation(startR, endR);
01103                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01104                         startAnimations.Add(animation);
01105                     }
01106                     else if (startAction is PathAction startP && endAction is PathAction endP)
01107                     {
01108                         IAnimation animation = new PathActionAnimation(startP, endP,
      linearisationResolution);
01109                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01110                         startAnimations.Add(animation);
01111                     }
01112                     else if (startAction is RasterImageAction startI && endAction is RasterImageAction
      endI)
01113                     {
01114                         IAnimation animation = new RasterImageActionAnimation(startI, endI);
01115                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01116                         startAnimations.Add(animation);
01117                     }
01118                     else if (startAction is FilteredGraphicsAction startF && endAction is
      FilteredGraphicsAction endF)
01119                     {
01120                         IAnimation animation = new FilteredGraphicsAnimation(startF, endF,
      linearisationResolution);
01121                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01122                         startAnimations.Add(animation);
01123                     }
01124                     else
01125                     {
01126                         startAnimations.Add(new ConstantAnimation<IGraphicsAction>(startAction));
01127                     }
```

```
01128                 }
01129                 else
01130                 {
01131                     startAnimations.Add(new ConstantAnimation<IGraphicsAction>(startAction));
01132                 }
01133             }
01134 
01135 
01136             List<IAnimation> endAnimations = new List<IAnimation>();
01137 
01138             for (int i = 0; i < this.EndValue.Actions.Count; i++)
01139             {
01140                 IGraphicsAction endAction = this.EndValue.Actions[i];
01141 
01142                 if (!string.IsNullOrEmpty(this.EndValue.Actions[i].Tag) &&
      startTaggedActions.TryGetValue(this.EndValue.Actions[i].Tag, out IGraphicsAction startAction) &&
      startAction.GetType() == endAction.GetType())
01143                 {
01144                     endAnimations.Add(computedAnimations[endAction.Tag]);
01145                 }
01146                 else
01147                 {
01148                     endAnimations.Add(new ConstantAnimation<IGraphicsAction>(endAction));
01149                 }
01150             }
01151 
01152             this.StartAnimations = startAnimations;
01153             this.EndAnimations = endAnimations;
01154         }
01155 
01156         public Graphics Interpolate(double position, Dictionary<string, IEasing> easings)
01157         {
01158             if (position <= 0)
01159             {
01160                 return this.StartValue;
01161             }
01162             else if (position >= 1)
01163             {
01164                 return this.EndValue;
01165             }
01166             else
01167             {
01168                 Graphics gpr = new Graphics();
01169 
01170                 if (position < 0.5)
01171                 {
01172                     for (int i = 0; i < StartAnimations.Count; i++)
01173                     {
01174                         gpr.Actions.Add(StartAnimations[i].Interpolate(position, easings));
01175                     }
01176                 }
01177                 else
01178                 {
01179                     for (int i = 0; i < EndAnimations.Count; i++)
01180                     {
01181                         gpr.Actions.Add(EndAnimations[i].Interpolate(position, easings));
01182                     }
01183                 }
01184 
01185                 return gpr;
01186             }
01187         }
01188     }
01189 
01190 /// <summary>
01191 /// A key frame for an animation.
01192 /// </summary>
01193     public class Frame
01194     {
01195 /// <summary>
01196 /// The duration of the frame, in milliseconds.
01197 /// </summary>
01198         public double Duration { get; }
01199 
01200 /// <summary>
01201 /// The contents of the frame.
01202 /// </summary>
01203         public Graphics Graphics { get; }
01204 
01205 /// <summary>
01206 /// Creates a new <see cref="Frame"/> with the specified contents and duration.
01207 /// </summary>
01208 /// <param name="graphics">The contents of the frame.</param>
01209 /// <param name="duration">The duration of the frame, in milliseconds.</param>
01210         public Frame(Graphics graphics, double duration)
01211         {
01212             this.Graphics = graphics;
```

```
01213             this.Duration = duration;
01214         }
01215     }
01216 
01217 /// <summary>
01218 /// Describes a function used to transform the transition speed.
01219 /// </summary>
01220     public interface IEasing
01221     {
01222 /// <summary>
01223 /// Applies the easing to the specified transition offset.
01224 /// </summary>
01225 /// <param name="value">The transition offset (ranging from 0 to 1).</param>
01226 /// <returns>The eased transition offset value.</returns>
01227         double Ease(double value);
01228     }
01229 
01230 /// <summary>
01231 /// Describes an easing defined by a Cubic Bezier curve.
01232 /// </summary>
01233     public class SplineEasing :  IEasing
01234     {
01235 /// <summary>
01236 /// The first control point of the curve.
01237 /// </summary>
01238         public Point ControlPoint1 { get; }
01239 
01240 /// <summary>
01241 /// The second control point of the curve.
01242 /// </summary>
01243         public Point ControlPoint2 { get; }
01244         private double[] SampledEasings { get; }
01245 
01246 /// <summary>
01247 /// Creates a new <see cref="SplineEasing"/> with the specified control points.  The start point is always (0, 0) and the end point is always (1, 1).
01248 /// </summary>
01249 /// <param name="controlPoint1">The first control point of the curve.  Both X and Y must be between 0 and 1, inclusive.</param>
01250 /// <param name="controlPoint2">The second control point of the curve.  Both X and Y must be between 0 and 1, inclusive.</param>
01251 /// <exception cref="ArgumentException">This exception is thrown if any coordinate of the control points is &lt; 0 or &gt; 1.</exception>
01252         public SplineEasing(Point controlPoint1, Point controlPoint2)
01253         {
01254             if (controlPoint1.X < 0 || controlPoint1.Y < 0 || controlPoint2.X > 1 || controlPoint2.Y > 1)
01255             {
01256                 throw new ArgumentException("The control point coordinates are out of range!  All coordinates must be within 0 and 1 (inclusive).");
01257             }
01258 
01259             this.ControlPoint1 = controlPoint1;
01260             this.ControlPoint2 = controlPoint2;
01261 
01262             Point[] pts = new Point[51];
01263 
01264             for (int i = 0; i <= 50; i++)
01265             {
01266                 double t = i / 50.0;
01267 
01268                 pts[i] = new Point(3 * (1 - t) * (1 - t) * t * controlPoint1.X + 3 * (1 - t) * t * t * controlPoint2.X + t * t * t, 3 * (1 - t) * (1 - t) * t * controlPoint1.Y + 3 * (1 - t) * t * t * controlPoint2.Y + t * t * t);
01269             }
01270 
01271             double[] sampledEasings = new double[51];
01272 
01273 
01274             double currX = 0;
01275             double currY = pts[0].Y;
01276 
01277             int nextSample = 1;
01278             sampledEasings[0] = pts[0].Y;
01279 
01280             for (int i = 1; i < sampledEasings.Length; i++)
01281             {
01282                 double targetX = nextSample / 50.0;
01283 
01284                 double newX = pts[i].X;
01285 
01286                 while (currX <= targetX && newX >= targetX)
01287                 {
01288                     sampledEasings[nextSample] = currY + (targetX - currX) / (newX - currX) * (pts[i].Y - currY);
01289                     nextSample++;
01290                     targetX = nextSample / 50.0;
```

```
01291                 }
01292
01293                 currX = newX;
01294                 currY = pts[i].Y;
01295             }
01296
01297             this.SampledEasings = sampledEasings;
01298         }
01299
01300 /// <inheritdoc/>
01301         public double Ease(double value)
01302         {
01303             if (value <= 0)
01304             {
01305                 return 0;
01306             }
01307             else if (value >= 1)
01308             {
01309                 return 1;
01310             }
01311             else
01312             {
01313                 int lowIndex = (int)Math.Floor(value * 50);
01314                 int highIndex = (int)Math.Ceiling(value * 50);
01315
01316                 return SampledEasings[lowIndex] + (SampledEasings[highIndex] -
      SampledEasings[lowIndex]) * (value * 50 - lowIndex);
01317             }
01318         }
01319     }
01320
01321 /// <summary>
01322 /// Describes a linear easing (i.e., no easing).
01323 /// </summary>
01324     public class LinearEasing :  IEasing
01325     {
01326 /// <summary>
01327 /// Creates a new <see cref="LinearEasing"/>.
01328 /// </summary>
01329         public LinearEasing()
01330         {
01331
01332         }
01333
01334 /// <inheritdoc/>
01335         public double Ease(double value) => value;
01336     }
01337
01338 /// <summary>
01339 /// Describes the transition between two successive <see cref="Frame"/>s.
01340 /// </summary>
01341     public class Transition
01342     {
01343 /// <summary>
01344 /// The duration of the transition, in milliseconds.
01345 /// </summary>
01346         public double Duration { get; }
01347
01348 /// <summary>
01349 /// The <see cref="IEasing"/> to apply to all elements for which another easing is not specified.  Set
      to null to use the default linear easing.
01350 /// </summary>
01351         public IEasing OverallEasing { get; } = null;
01352
01353 /// <summary>
01354 /// A dictionary associating graphic action tags to the corresponding <see cref="IEasing"/>.
01355 /// </summary>
01356         public Dictionary<string, IEasing> Easings { get; } = null;
01357
01358 /// <summary>
01359 /// Creates a new <see cref="Transition"/> with the specified duration and easings.
01360 /// </summary>
01361 /// <param name="duration">The duration of the transition, in milliseconds.</param>
01362 /// <param name="easing">The <see cref="IEasing"/> to apply to all elements for which another easing
      is not specified.  Set to null to use the default linear easing.</param>
01363 /// <param name="easings">A dictionary associating graphic action tags to the corresponding <see
      cref="IEasing"/>.</param>
01364         public Transition(double duration, IEasing easing = null, Dictionary<string, IEasing> easings
      = null)
01365         {
01366             if (easing == null)
01367             {
01368                 easing = new LinearEasing();
01369             }
01370
01371             this.Duration = duration;
01372             this.OverallEasing = easing;
```

```
01373             this.Easings = easings;
01374         }
01375     }
01376 
01377 /// <summary>
01378 /// Describes an animation constituted by a number of frames and transitions between them.
01379 /// </summary>
01380     public class Animation
01381     {
01382 /// <summary>
01383 /// The key frames of the animation.
01384 /// </summary>
01385         public ImmutableList<Frame> Frames { get; private set; } = ImmutableList<Frame>.Empty;
01386 
01387 /// <summary>
01388 /// The transitions between successive frames of the animation.  This array always contains one fewer element than <see cref="Frames"/>.
01389 /// </summary>
01390         public ImmutableList<Transition> Transitions { get; private set; } = ImmutableList<Transition>.Empty;
01391 
01392         private ImmutableList<GraphicsAnimation> Animations { get; set; } = ImmutableList<GraphicsAnimation>.Empty;
01393 
01394 
01395 /// <summary>
01396 /// The width of the animation.
01397 /// </summary>
01398         public double Width { get; set; }
01399 
01400 /// <summary>
01401 /// The height of the animation.
01402 /// </summary>
01403         public double Height { get; set; }
01404 
01405 /// <summary>
01406 /// The background colour of the animation.
01407 /// </summary>
01408         public Colour Background { get; set; } = Colour.FromRgba(255, 255, 255, 0);
01409 
01410 /// <summary>
01411 /// The absolute length between successive samples to use when linearising <see cref="GraphicsPath"/>s.
01412 /// </summary>
01413         public double LinearisationResolution { get; }
01414 
01415 /// <summary>
01416 /// The total duration of the animation (not including the number of repeats).
01417 /// </summary>
01418         public double Duration { get; private set; } = 0;
01419 
01420 /// <summary>
01421 /// The number of times that the animation should repeat.
01422 /// </summary>
01423         public int RepeatCount { get; set; } = 0;
01424 
01425 /// <summary>
01426 /// Creates a new <see cref="Animation"/> with the specified width, height and linearisation resolution.
01427 /// </summary>
01428 /// <param name="width">The width of the animation.</param>
01429 /// <param name="height">The height of the animation.</param>
01430 /// <param name="linearisationResolution">The absolute length between successive samples to use when linearising <see cref="GraphicsPath"/>s.</param>
01431         public Animation(double width, double height, double linearisationResolution)
01432         {
01433             this.Width = width;
01434             this.Height = height;
01435             this.LinearisationResolution = linearisationResolution;
01436         }
01437 
01438 /// <summary>
01439 /// Obtains the (interpolated) frame that should be displayed after the specified time has passed since the start of the animation.
01440 /// </summary>
01441 /// <param name="time">The time since the start of the animation (in milliseconds).</param>
01442 /// <returns>A <see cref="Page"/> containing the interpolated frame that should be displayed after the specified time has passed since the start of the animation.</returns>
01443         public Page GetFrameAtAbsolute(double time)
01444         {
01445             if (time > Duration)
01446             {
01447                 if (this.RepeatCount <= 0)
01448                 {
01449                     time = time % this.Duration;
01450                 }
01451                 else
```

```
01452                 {
01453                     time = Math.Max(time % this.Duration, time - this.Duration * RepeatCount);
01454                 }
01455             }
01456 
01457             if (time <= 0)
01458             {
01459                 Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01460                 pag.Graphics.DrawGraphics(0, 0, this.Frames[0].Graphics);
01461                 return pag;
01462             }
01463             else if (time > this.Duration)
01464             {
01465                 Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01466                 pag.Graphics.DrawGraphics(0, 0, this.Frames[this.Frames.Count - 1].Graphics);
01467                 return pag;
01468             }
01469             else
01470             {
01471                 double currTime = 0;
01472 
01473                 for (int i = 0; i < Frames.Count; i++)
01474                 {
01475                     if (i > 0)
01476                     {
01477                         double newTime = currTime + Transitions[i - 1].Duration;
01478 
01479                         if (currTime <= time && newTime >= time)
01480                         {
01481                             Frame previousFrame = Frames[i - 1];
01482 
01483                             Frame nextFrame = Frames[i];
01484 
01485                             if (previousFrame != null && nextFrame != null)
01486                             {
01487                                 double position = Transitions[i - 1].OverallEasing.Ease((time -
      currTime) / (newTime - currTime));
01488 
01489                                 Graphics gpr = Animations[i - 1].Interpolate(position, Transitions[i -
      1].Easings);
01490                                 Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01491                                 pag.Graphics.DrawGraphics(0, 0, gpr);
01492                                 return pag;
01493                             }
01494                             else if (previousFrame == null && nextFrame != null)
01495                             {
01496                                 Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01497                                 pag.Graphics.DrawGraphics(0, 0, nextFrame.Graphics);
01498                                 return pag;
01499                             }
01500                             else if (previousFrame != null && nextFrame == null)
01501                             {
01502                                 Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01503                                 pag.Graphics.DrawGraphics(0, 0, previousFrame.Graphics);
01504                                 return pag;
01505                             }
01506                             else
01507                             {
01508                                 Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01509                                 return pag;
01510                             }
01511                         }
01512 
01513                         currTime = newTime;
01514                     }
01515 
01516 
01517                     {
01518                         double newTime = currTime + Frames[i].Duration;
01519 
01520                         if (currTime <= time && newTime >= time)
01521                         {
01522                             Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01523                             pag.Graphics.DrawGraphics(0, 0, Frames[i].Graphics);
01524                             return pag;
01525                         }
01526 
01527                         currTime = newTime;
01528                     }
01529 
01530                 }
01531 
```

```
01532
01533                 {
01534                     Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01535                     pag.Graphics.DrawGraphics(0, 0, this.Frames[this.Frames.Count - 1].Graphics);
01536                     return pag;
01537                 }
01538             }
01539         }
01540
01541 /// <summary>
01542 /// Obtains the (interpolated) frame that should be displayed after the specified relative time has
      passed since the start of the animation.
01543 /// </summary>
01544 /// <param name="relativeTime">The time since the start of the animation (ranging from 0 for the start
      of the animation, to 1 for the end of the animation).</param>
01545 /// <returns>A <see cref="Page"/> containing the interpolated frame that should be displayed after the
      specified time has passed since the start of the animation.</returns>
01546         public Page GetFrameAtRelative(double relativeTime)
01547         {
01548             return GetFrameAtAbsolute(this.Duration * relativeTime);
01549         }
01550
01551 /// <summary>
01552 /// Adds a new frame to the animation, with the specified transition.
01553 /// </summary>
01554 /// <param name="frame">The new frame to add to the animation.</param>
01555 /// <param name="transition">The transition that should be applied between the previous frame and the
      new frame.  This parameter is ignored for the first frame.  If this is <see langword="null"/>, the
      animation will abruptly change from one frame to the next.</param>
01556         public void AddFrame(Frame frame, Transition transition = null)
01557         {
01558             this.Frames = this.Frames.Add(frame);
01559             this.Duration += frame.Duration;
01560
01561             if (this.Frames.Count > 1)
01562             {
01563                 if (transition == null)
01564                 {
01565                     transition = new Transition(0, new LinearEasing());
01566                 }
01567
01568                 this.Transitions = this.Transitions.Add(transition);
01569                 this.Animations = this.Animations.Add(new
      GraphicsAnimation(this.Frames[this.Frames.Count - 2].Graphics, this.Frames[this.Frames.Count -
      1].Graphics, this.LinearisationResolution));
01570                 this.Duration += transition.Duration;
01571             }
01572         }
01573
01574 /// <summary>
01575 /// Removes the last frame from the animation (and the corresponding transition).
01576 /// </summary>
01577 /// <exception cref="ArgumentOutOfRangeException">Thrown if there are no frames in the
      animation.</exception>
01578         public void RemoveLastFrame()
01579         {
01580             if (this.Frames.Count > 0)
01581             {
01582                 if (this.Frames.Count > 1)
01583                 {
01584                     double lostDuration = this.Frames[this.Frames.Count - 1].Duration +
      this.Transitions[this.Transitions.Count - 1].Duration;
01585
01586                     this.Frames = this.Frames.RemoveAt(this.Frames.Count - 1);
01587                     this.Animations = this.Animations.RemoveAt(this.Animations.Count - 1);
01588                     this.Transitions = this.Transitions.RemoveAt(this.Transitions.Count - 1);
01589
01590                     this.Duration -= lostDuration;
01591                 }
01592                 else
01593                 {
01594                     this.Frames = ImmutableList<Frame>.Empty;
01595                     this.Duration = 0;
01596                 }
01597             }
01598             else
01599             {
01600                 throw new ArgumentOutOfRangeException("There are no frames in the animation!");
01601             }
01602         }
01603     }
01604 }
```

8.51 Brush.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.Collections.Immutable;
00022 using System.Linq;
00023 using System.Text;
00024
00025 namespace VectSharp
00026 {
00027 /// <summary>
00028 /// Represents a brush used to fill or stroke graphics elements.  This could be a solid colour, or a
      more complicated gradient or pattern.
00029 /// </summary>
00030     public abstract class Brush
00031     {
00032         internal Brush() { }
00033
00034 /// <summary>
00035 /// Returns a brush corresponding the current instance, with the specified <paramref name="opacity"/>
      multiplication applied.
00036 /// </summary>
00037 /// <param name="opacity">The value that will be used to multiply the opacity of the brush.</param>
00038 /// <returns>A brush corresponding the current instance, with the specified <paramref name="opacity"/>
      multiplication applied.</returns>
00039         public abstract Brush MultiplyOpacity(double opacity);
00040
00041 /// <summary>
00042 /// Implicitly converts a <see cref="Colour"/> into a <see cref="SolidColourBrush"/>.
00043 /// </summary>
00044 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00045         public static implicit operator Brush(Colour colour)
00046         {
00047             return new SolidColourBrush(colour);
00048         }
00049     }
00050
00051 /// <summary>
00052 /// Represents a brush painting with a single solid colour.
00053 /// </summary>
00054     public class SolidColourBrush :  Brush
00055     {
00056 /// <summary>
00057 /// The colour of the brush.
00058 /// </summary>
00059         public Colour Colour { get; }
00060
00061 /// <summary>
00062 /// Red component of the colour.  Range:  [0, 1].
00063 /// </summary>
00064         public double R => Colour.R;
00065
00066 /// <summary>
00067 /// Green component of the colour.  Range:  [0, 1].
00068 /// </summary>
00069         public double G => Colour.G;
00070
00071 /// <summary>
00072 /// Blue component of the colour.  Range:  [0, 1].
00073 /// </summary>
00074         public double B => Colour.B;
00075
00076 /// <summary>
00077 /// Alpha component of the colour.  Range:  [0, 1].
00078 /// </summary>
00079         public double A => Colour.A;
00080
00081 /// <summary>
00082 /// Creates a new <see cref="SolidColourBrush"/> with the specified <paramref name="colour"/>.
```

```
00083 /// </summary>
00084 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00085         public SolidColourBrush(Colour colour)
00086         {
00087             this.Colour = colour;
00088         }
00089 
00090 /// <inheritdoc/>
00091         public override Brush MultiplyOpacity(double opacity)
00092         {
00093             return new SolidColourBrush(this.Colour.WithAlpha(this.Colour.A * opacity));
00094         }
00095 
00096 /// <summary>
00097 /// Implicitly converts a <see cref="Colour"/> into a <see cref="SolidColourBrush"/>.
00098 /// </summary>
00099 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00100         public static implicit operator SolidColourBrush(Colour colour)
00101         {
00102             return new SolidColourBrush(colour);
00103         }
00104     }
00105 
00106 /// <summary>
00107 /// Represents a colour stop in a gradient.
00108 /// </summary>
00109     public struct GradientStop
00110     {
00111 /// <summary>
00112 /// The <see cref="Colour"/> at the gradient stop.
00113 /// </summary>
00114         public Colour Colour { get; }
00115 
00116 /// <summary>
00117 /// The offset of the gradient stop.  Range:  [0, 1].
00118 /// </summary>
00119         public double Offset { get; }
00120 
00121 /// <summary>
00122 /// Creates a new <see cref="GradientStop"/> instance.
00123 /// </summary>
00124 /// <param name="colour">The <see cref="Colour"/> at the gradient stop.</param>
00125 /// <param name="offset">The offset of the gradient stop.  Range:  [0, 1].</param>
00126         public GradientStop(Colour colour, double offset)
00127         {
00128             this.Colour = colour;
00129             this.Offset = Math.Max(0, Math.Min(1, offset));
00130         }
00131 
00132 /// <summary>
00133 /// Returns a <see cref="GradientStop"/> corresponding to the current instance, whose colour's opacity
      has been multiplied by the specified value.
00134 /// </summary>
00135 /// <param name="opacity">The value that will be used to multiply the colour's opacity.</param>
00136 /// <returns>A <see cref="GradientStop"/> corresponding to the current instance, whose colour's
      opacity has been multiplied by the specified value.</returns>
00137         public GradientStop MultiplyOpacity(double opacity)
00138         {
00139             return new GradientStop(this.Colour.WithAlpha(this.Colour.A * opacity), this.Offset);
00140         }
00141     }
00142 
00143 /// <summary>
00144 /// Represents a read-only list of <see cref="GradientStop"/>s.
00145 /// </summary>
00146     public class GradientStops :  IReadOnlyList<GradientStop>
00147     {
00148 /// <summary>
00149 /// The minimum distance that is enforced between consecutive gradient stops.
00150 /// </summary>
00151         public static readonly double StopTolerance = 1e-7;
00152 
00153 /// <inheritdoc/>
00154         public GradientStop this[int index] => gradientStops[index];
00155 
00156 /// <inheritdoc/>
00157         public int Count => gradientStops.Count;
00158 
00159         private ImmutableList<GradientStop> gradientStops { get; set; }
00160 
00161 /// <inheritdoc/>
00162         public IEnumerator<GradientStop> GetEnumerator()
00163         {
00164             return ((IEnumerable<GradientStop>)gradientStops).GetEnumerator();
00165         }
00166 
00167         IEnumerator IEnumerable.GetEnumerator()
```

```
00168         {
00169             return ((IEnumerable)gradientStops).GetEnumerator();
00170         }
00171 
00172 /// <summary>
00173 /// Creates a new <see cref="GradientStops"/> instance containing the specified gradient stops.
00174 /// </summary>
00175 /// <param name="gradientStops">The gradient stops that will be contained in the <see
      cref="GradientStops"/> object.</param>
00176         public GradientStops(IEnumerable<GradientStop> gradientStops)
00177         {
00178             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00179 
00180             if (stops.Count == 0)
00181             {
00182                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00183             }
00184 
00185             if (stops[0].Offset > 0)
00186             {
00187                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00188             }
00189 
00190             if (stops[stops.Count - 1].Offset < 1)
00191             {
00192                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00193             }
00194 
00195             for (int i = 1; i < stops.Count - 1; i++)
00196             {
00197                 bool closeToPrevious = (stops[i].Offset - stops[i - 1].Offset < StopTolerance);
00198                 bool closeToNext = (stops[i + 1].Offset - stops[i].Offset < StopTolerance);
00199 
00200                 if (closeToPrevious && !closeToNext)
00201                 {
00202                     stops[i] = new GradientStop(stops[i].Colour, stops[i - 1].Offset + StopTolerance);
00203                 }
00204                 else if (!closeToPrevious && closeToNext)
00205                 {
00206                     stops[i] = new GradientStop(stops[i].Colour, stops[i + 1].Offset - StopTolerance);
00207                 }
00208                 else if (closeToPrevious && closeToNext)
00209                 {
00210                     stops.RemoveAt(i);
00211                     i--;
00212                 }
00213             }
00214 
00215             this.gradientStops = ImmutableList.Create(stops.ToArray());
00216         }
00217 
00218 /// <summary>
00219 /// Creates a new <see cref="GradientStops"/> instance containing the specified gradient stops.
00220 /// </summary>
00221 /// <param name="gradientStops">The gradient stops that will be contained in the <see
      cref="GradientStops"/> object.</param>
00222         public GradientStops(params GradientStop[] gradientStops) :
      this((IEnumerable<GradientStop>)gradientStops)
00223         {
00224 
00225         }
00226 
00227 /// <summary>
00228 /// Gets the colour at a certain position on the gradient.
00229 /// </summary>
00230 /// <param name="position">The position in the gradient (ranging from 0 to 1).</param>
00231 /// <returns>The colour of the gradient at the specified position.</returns>
00232         public Colour GetColourAt(double position)
00233         {
00234             if (position <= 0)
00235             {
00236                 return this.gradientStops[0].Colour;
00237             }
00238             else if (position >= 1)
00239             {
00240                 return this.gradientStops[this.gradientStops.Count - 1].Colour;
00241             }
00242             else
00243             {
00244                 for (int i = 1; i < this.gradientStops.Count; i++)
00245                 {
00246                     if (this.gradientStops[i - 1].Offset < position && this.gradientStops[i].Offset >=
      position)
00247                     {
00248                         double factor = (position - this.gradientStops[i - 1].Offset) /
      (this.gradientStops[i].Offset - this.gradientStops[i - 1].Offset);
```

```
00249
00250                         return Colour.FromRgba(this.gradientStops[i - 1].Colour.R * (1 - factor) +
     this.gradientStops[i].Colour.R * factor,
00251                             this.gradientStops[i - 1].Colour.G * (1 - factor) +
     this.gradientStops[i].Colour.G * factor,
00252                             this.gradientStops[i - 1].Colour.B * (1 - factor) +
     this.gradientStops[i].Colour.B * factor,
00253                             this.gradientStops[i - 1].Colour.A * (1 - factor) +
     this.gradientStops[i].Colour.A * factor);
00254                     }
00255                 }
00256                 return Colour.FromRgba(0, 0, 0, 0);
00257             }
00258         }
00259
00260 /// <summary>
00261 /// Convert a <see cref="GradientStops"/> object to a function that assigns a colour to values between
     0 and 1.
00262 /// </summary>
00263 /// <param name="stops">The <see cref="GradientStops"/> to convert.</param>
00264         public static implicit operator Func<double, Colour>(GradientStops stops) =>
     stops.GetColourAt;
00265     }
00266
00267 /// <summary>
00268 /// Represents a brush painting with a gradient.
00269 /// </summary>
00270     public abstract class GradientBrush :  Brush
00271     {
00272 /// <summary>
00273 /// The colour stops in the gradient.
00274 /// </summary>
00275         public GradientStops GradientStops { get; protected internal set; }
00276
00277         internal GradientBrush() { }
00278     }
00279
00280 /// <summary>
00281 /// Represents a brush painting with a linear gradient.
00282 /// </summary>
00283     public class LinearGradientBrush :  GradientBrush
00284     {
00285 /// <summary>
00286 /// The starting point of the gradient.  Note that this is relative to the current coordinate system
     when the gradient is used.
00287 /// </summary>
00288         public Point StartPoint { get; }
00289
00290 /// <summary>
00291 /// The end point of the gradient.  Note that this is relative to the current coordinate system when
     the gradient is used.
00292 /// </summary>
00293         public Point EndPoint { get; }
00294
00295 /// <summary>
00296 /// Creates a new <see cref="LinearGradientBrush"/> with the specified start point, end point and
     gradient stops.
00297 /// </summary>
00298 /// <param name="startPoint">The starting point of the gradient.  Note that this is relative to the
     current coordinate system when the gradient is used.</param>
00299 /// <param name="endPoint">The ending point of the gradient.  Note that this is relative to the
     current coordinate system when the gradient is used.</param>
00300 /// <param name="gradientStops">The colour stops in the gradient.</param>
00301         public LinearGradientBrush(Point startPoint, Point endPoint, IEnumerable<GradientStop>
     gradientStops)
00302         {
00303             this.StartPoint = startPoint;
00304             this.EndPoint = endPoint;
00305
00306             this.GradientStops = new GradientStops(gradientStops);
00307         }
00308
00309 /// <summary>
00310 /// Creates a new <see cref="LinearGradientBrush"/> with the specified start point, end point and
     gradient stops.
00311 /// </summary>
00312 /// <param name="startPoint">The starting point of the gradient.  Note that this is relative to the
     current coordinate system when the gradient is used.</param>
00313 /// <param name="endPoint">The ending point of the gradient.  Note that this is relative to the
     current coordinate system when the gradient is used.</param>
00314 /// <param name="gradientStops">The colour stops in the gradient.</param>
00315         public LinearGradientBrush(Point startPoint, Point endPoint, params GradientStop[]
     gradientStops)
00316         {
00317             this.StartPoint = startPoint;
00318             this.EndPoint = endPoint;
00319             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
```

```
el).ToList();
00320
00321             if (stops.Count == 0)
00322             {
00323                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00324             }
00325
00326             if (stops[0].Offset > 0)
00327             {
00328                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00329             }
00330
00331             if (stops[stops.Count - 1].Offset < 1)
00332             {
00333                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00334             }
00335
00336             this.GradientStops = new GradientStops(gradientStops);
00337         }
00338
00339 /// <summary>
00340 /// Returns a <see cref="LinearGradientBrush"/> with the same gradient stops as the current instance,
    whose start and end point correspond to the points of the current instance in the
00341 /// original reference frame of the <paramref name="referenceGraphics"/>.  This involves computing the
    current transform matrix of the <paramref name="referenceGraphics" />, inverting it,
00342 /// and applying the inverse matrix to the <see cref="StartPoint"/> and <see cref="EndPoint"/> of the
    current instance.
00343 /// </summary>
00344 /// <param name="referenceGraphics">The <see cref="Graphics"/> whose original reference frame is to be
    used.</param>
00345 /// <returns>A <see cref="LinearGradientBrush"/> with the same gradient stops as the current instance,
    whose start and end point correspond to the points of the current instance in the
00346 /// original reference frame of the <paramref name="referenceGraphics"/>.</returns>
00347         public LinearGradientBrush RelativeTo(Graphics referenceGraphics)
00348         {
00349             Stack<double[,]> transformMatrix = new Stack<double[,]>();
00350             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00351
00352             for (int i = 0; i < referenceGraphics.Actions.Count; i++)
00353             {
00354                 if (referenceGraphics.Actions[i] is TransformAction)
00355                 {
00356                     TransformAction trf = referenceGraphics.Actions[i] as TransformAction;
00357
00358                     if (trf.Delta != null)
00359                     {
00360                         currMatrix = Graphics.Multiply(currMatrix,
    Graphics.TranslationMatrix(trf.Delta.Value.X, trf.Delta.Value.Y));
00361                     }
00362                     else if (trf.Angle != null)
00363                     {
00364                         currMatrix = Graphics.Multiply(currMatrix,
    Graphics.RotationMatrix(trf.Angle.Value));
00365                     }
00366                     else if (trf.Scale != null)
00367                     {
00368                         currMatrix = Graphics.Multiply(currMatrix,
    Graphics.ScaleMatrix(trf.Scale.Value.Width, trf.Scale.Value.Height));
00369                     }
00370                     else if (trf.Matrix != null)
00371                     {
00372                         currMatrix = Graphics.Multiply(currMatrix, trf.Matrix);
00373                     }
00374                 }
00375                 else if (referenceGraphics.Actions[i] is StateAction)
00376                 {
00377                     if (((StateAction)referenceGraphics.Actions[i]).StateActionType ==
    StateAction.StateActionTypes.Save)
00378                     {
00379                         transformMatrix.Push(currMatrix);
00380                     }
00381                     else
00382                     {
00383                         currMatrix = transformMatrix.Pop();
00384                     }
00385                 }
00386             }
00387
00388             currMatrix = Graphics.Invert(currMatrix);
00389
00390             Point p1 = Graphics.Multiply(currMatrix, this.StartPoint);
00391             Point p2 = Graphics.Multiply(currMatrix, this.EndPoint);
00392
00393             return new LinearGradientBrush(p1, p2, this.GradientStops);
00394         }
00395
00396 /// <inheritdoc/>
```

```
00397         public override Brush MultiplyOpacity(double opacity)
00398         {
00399             return new LinearGradientBrush(this.StartPoint, this.EndPoint, from el in
      this.GradientStops select el.MultiplyOpacity(opacity));
00400         }
00401     }
00402
00403 /// <summary>
00404 /// Represents a brush painting with a radial gradient.
00405 /// </summary>
00406     public class RadialGradientBrush :  GradientBrush
00407     {
00408 /// <summary>
00409 /// The focal point of the gradient (i.e.  the point within the circle where the gradient starts).
00410 /// </summary>
00411         public Point FocalPoint { get; }
00412
00413 /// <summary>
00414 /// Represents the centre of the gradient.
00415 /// </summary>
00416         public Point Centre { get; }
00417
00418 /// <summary>
00419 /// The radius of the gradient.
00420 /// </summary>
00421         public double Radius { get; }
00422
00423 /// <summary>
00424 /// Creates a new <see cref="RadialGradientBrush"/> with the specified focal point, centre, radius and
      gradient stops.
00425 /// </summary>
00426 /// <param name="focalPoint">The focal point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00427 /// <param name="centre">The centre of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00428 /// <param name="radius">The radius of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00429 /// <param name="gradientStops">The colour stops in the gradient.</param>
00430         public RadialGradientBrush(Point focalPoint, Point centre, double radius, params
      GradientStop[] gradientStops)
00431         {
00432             if (new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Modulus() > radius)
00433             {
00434                 Point norm = new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Normalize();
00435                 focalPoint = new Point(centre.X + norm.X * radius, centre.Y + norm.Y * radius);
00436             }
00437
00438             this.FocalPoint = focalPoint;
00439             this.Centre = centre;
00440             this.Radius = radius;
00441
00442             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00443
00444             if (stops.Count == 0)
00445             {
00446                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00447             }
00448
00449             if (stops[0].Offset > 0)
00450             {
00451                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00452             }
00453
00454             if (stops[stops.Count - 1].Offset < 1)
00455             {
00456                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00457             }
00458
00459             this.GradientStops = new GradientStops(gradientStops);
00460         }
00461
00462 /// <summary>
00463 /// Creates a new <see cref="RadialGradientBrush"/> with the specified focal point, centre, radius and
      gradient stops.
00464 /// </summary>
00465 /// <param name="focalPoint">The focal point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00466 /// <param name="centre">The centre of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00467 /// <param name="radius">The radius of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00468 /// <param name="gradientStops">The colour stops in the gradient.</param>
00469         public RadialGradientBrush(Point focalPoint, Point centre, double radius,
      IEnumerable<GradientStop> gradientStops)
00470         {
00471             if (new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Modulus() > radius)
```

```
00472             {
00473                 Point norm = new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Normalize();
00474                 focalPoint = new Point(centre.X + norm.X * radius, centre.Y + norm.Y * radius);
00475             }
00476 
00477             this.FocalPoint = focalPoint;
00478             this.Centre = centre;
00479             this.Radius = radius;
00480 
00481             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00482 
00483             if (stops.Count == 0)
00484             {
00485                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00486             }
00487 
00488             if (stops[0].Offset > 0)
00489             {
00490                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00491             }
00492 
00493             if (stops[stops.Count - 1].Offset < 1)
00494             {
00495                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00496             }
00497 
00498             this.GradientStops = new GradientStops(gradientStops);
00499         }
00500 
00501 /// <inheritdoc/>
00502         public override Brush MultiplyOpacity(double opacity)
00503         {
00504             return new RadialGradientBrush(this.FocalPoint, this.Centre, this.Radius, from el in
      this.GradientStops select el.MultiplyOpacity(opacity));
00505         }
00506     }
00507 }
```

8.52 Colour.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 
00020 namespace VectSharp
00021 {
00022 /// <summary>
00023 /// Represents an RGB colour.
00024 /// </summary>
00025     public partial struct Colour :  IEquatable<Colour>
00026     {
00027 /// <summary>
00028 /// Red component of the colour.  Range:  [0, 1].
00029 /// </summary>
00030         public double R;
00031 
00032 /// <summary>
00033 /// Green component of the colour.  Range:  [0, 1].
00034 /// </summary>
00035         public double G;
00036 
00037 /// <summary>
00038 /// Blue component of the colour.  Range:  [0, 1].
00039 /// </summary>
00040         public double B;
00041 
00042 /// <summary>
```

```
00043 /// Alpha component of the colour.  Range:  [0, 1].
00044 /// </summary>
00045         public double A;
00046 
00047         private Colour(double r, double g, double b, double a)
00048         {
00049             R = r;
00050             G = g;
00051             B = b;
00052             A = a;
00053         }
00054 
00055 /// <summary>
00056 /// Create a new colour from RGB (red, green and blue) values.
00057 /// </summary>
00058 /// <param name="r">The red component of the colour.  Range:  [0, 1].</param>
00059 /// <param name="g">The green component of the colour.  Range:  [0, 1].</param>
00060 /// <param name="b">The blue component of the colour.  Range:  [0, 1].</param>
00061 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00062         public static Colour FromRgb(double r, double g, double b)
00063         {
00064             return new Colour(r, g, b, 1);
00065         }
00066 
00067 /// <summary>
00068 /// Create a new colour from RGB (red, green and blue) values.
00069 /// </summary>
00070 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00071 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00072 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00073 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00074         public static Colour FromRgb(byte r, byte g, byte b)
00075         {
00076             return new Colour(r / 255.0, g / 255.0, b / 255.0, 1);
00077         }
00078 
00079 /// <summary>
00080 /// Create a new colour from RGB (red, green and blue) values.
00081 /// </summary>
00082 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00083 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00084 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00085 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00086         public static Colour FromRgb(int r, int g, int b)
00087         {
00088             return new Colour(r / 255.0, g / 255.0, b / 255.0, 1);
00089         }
00090 
00091 /// <summary>
00092 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00093 /// </summary>
00094 /// <param name="r">The red component of the colour.  Range:  [0, 1].</param>
00095 /// <param name="g">The green component of the colour.  Range:  [0, 1].</param>
00096 /// <param name="b">The blue component of the colour.  Range:  [0, 1].</param>
00097 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00098 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00099         public static Colour FromRgba(double r, double g, double b, double a)
00100         {
00101             return new Colour(r, g, b, a);
00102         }
00103 
00104 /// <summary>
00105 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00106 /// </summary>
00107 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00108 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00109 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00110 /// <param name="a">The alpha component of the colour.  Range:  [0, 255].</param>
00111 /// <returns>A <see cref="Colour"/><see cref="Colour"/> struct with the specified
      components.</returns>
00112         public static Colour FromRgba(byte r, byte g, byte b, byte a)
00113         {
00114             return new Colour(r / 255.0, g / 255.0, b / 255.0, a / 255.0);
00115         }
00116 
00117 /// <summary>
00118 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00119 /// </summary>
00120 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00121 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00122 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00123 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00124 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00125         public static Colour FromRgba(byte r, byte g, byte b, double a)
```

```
00126         {
00127             return new Colour(r / 255.0, g / 255.0, b / 255.0, a);
00128         }
00129 /// <summary>
00130 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00131 /// </summary>
00132 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00133 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00134 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00135 /// <param name="a">The alpha component of the colour.  Range:  [0, 255].</param>
00136 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00137         public static Colour FromRgba(int r, int g, int b, int a)
00138         {
00139             return new Colour(r / 255.0, g / 255.0, b / 255.0, a / 255.0);
00140         }
00141 
00142 /// <summary>
00143 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00144 /// </summary>
00145 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00146 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00147 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00148 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00149 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00150         public static Colour FromRgba(int r, int g, int b, double a)
00151         {
00152             return new Colour(r / 255.0, g / 255.0, b / 255.0, a);
00153         }
00154 
00155 /// <summary>
00156 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00157 /// </summary>
00158 /// <param name="colour">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00159 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00160         public static Colour FromRgba((int r, int g, int b, double a) colour)
00161         {
00162             return new Colour(colour.r / 255.0, colour.g / 255.0, colour.b / 255.0, colour.a);
00163         }
00164 
00165 /// <inheritdoc/>
00166         public override bool Equals(object obj)
00167         {
00168             if (!(obj is Colour))
00169             {
00170                 return false;
00171             }
00172             else
00173             {
00174                 return this.Equals((Colour)obj);
00175             }
00176         }
00177 
00178 /// <inheritdoc/>
00179         public bool Equals(Colour col)
00180         {
00181             return col.R == this.R && col.G == this.G && col.B == this.B && col.A == this.A;
00182         }
00183 
00184 /// <inheritdoc/>
00185         public static bool operator ==(Colour col1, Colour col2)
00186         {
00187             return col1.R == col2.R && col1.G == col2.G && col1.B == col2.B && col1.A == col2.A;
00188         }
00189 
00190 /// <inheritdoc/>
00191         public static bool operator !=(Colour col1, Colour col2)
00192         {
00193             return col1.R != col2.R || col1.G != col2.G || col1.B != col2.B || col1.A != col2.A;
00194         }
00195 
00196 /// <inheritdoc/>
00197         public override int GetHashCode()
00198         {
00199             return (int)(this.R * 255 + this.G * 255 * 255 + this.B * 255 * 255 * 255 + this.A * 255 *
      255 * 255 * 255);
00200         }
00201 
00202 /// <summary>
00203 /// Convert the <see cref="Colour"/> object into a hex string that is constituted by a "#" followed by
      two-digit hexadecimal representations of the red, green and blue components of the colour (in the
      range 0x00 - 0xFF).
00204 /// Optionally also includes opacity (alpha channel) data.
00205 /// </summary>
00206 /// <param name="includeAlpha">Whether two additional hex digits representing the colour's opacity
      (alpha channel) should be included in the string.</param>
00207 /// <returns>A hex colour string.</returns>
```

```
00208         public string ToCSSString(bool includeAlpha)
00209         {
00210             if (includeAlpha)
00211             {
00212                 return "#" + ((int)Math.Round(this.R * 255)).ToString("X2") + ((int)Math.Round(this.G * 255)).ToString("X2") + ((int)Math.Round(this.B * 255)).ToString("X2") + ((int)Math.Round(this.A * 255)).ToString("X2");
00213             }
00214             else
00215             {
00216                 return "#" + ((int)Math.Round(this.R * 255)).ToString("X2") + ((int)Math.Round(this.G * 255)).ToString("X2") + ((int)Math.Round(this.B * 255)).ToString("X2");
00217             }
00218         }
00219 
00220 /// <summary>
00221 /// Convert a CSS colour string into a <see cref="Colour"/> object.
00222 /// </summary>
00223 /// <param name="cssString">The CSS colour string.  In addition to 148 standard colour names (case-insensitive), #RGB, #RGBA, #RRGGBB and #RRGGBBAA hex strings and rgb(r, g, b) and rgba(r, g, b, a) functional colour notations are supported.</param>
00224 /// <returns></returns>
00225         public static Colour?  FromCSSString(string cssString)
00226         {
00227             if (cssString.StartsWith("#"))
00228             {
00229                 cssString = cssString.Substring(1);
00230 
00231                 try
00232                 {
00233                     if (cssString.Length == 3)
00234                     {
00235                         byte r = byte.Parse(cssString.Substring(0, 1) + cssString.Substring(0, 1), System.Globalization.NumberStyles.HexNumber);
00236                         byte g = byte.Parse(cssString.Substring(1, 1) + cssString.Substring(1, 1), System.Globalization.NumberStyles.HexNumber);
00237                         byte b = byte.Parse(cssString.Substring(2, 1) + cssString.Substring(2, 1), System.Globalization.NumberStyles.HexNumber);
00238 
00239                         return Colour.FromRgb(r, g, b);
00240                     }
00241                     else if (cssString.Length == 4)
00242                     {
00243                         byte r = byte.Parse(cssString.Substring(0, 1) + cssString.Substring(0, 1), System.Globalization.NumberStyles.HexNumber);
00244                         byte g = byte.Parse(cssString.Substring(1, 1) + cssString.Substring(1, 1), System.Globalization.NumberStyles.HexNumber);
00245                         byte b = byte.Parse(cssString.Substring(2, 1) + cssString.Substring(2, 1), System.Globalization.NumberStyles.HexNumber);
00246                         byte a = byte.Parse(cssString.Substring(3, 1) + cssString.Substring(3, 1), System.Globalization.NumberStyles.HexNumber);
00247 
00248                         return Colour.FromRgba(r, g, b, a);
00249                     }
00250                     else if (cssString.Length == 6)
00251                     {
00252                         byte r = byte.Parse(cssString.Substring(0, 2), System.Globalization.NumberStyles.HexNumber);
00253                         byte g = byte.Parse(cssString.Substring(2, 2), System.Globalization.NumberStyles.HexNumber);
00254                         byte b = byte.Parse(cssString.Substring(4, 2), System.Globalization.NumberStyles.HexNumber);
00255 
00256                         return Colour.FromRgb(r, g, b);
00257                     }
00258                     else if (cssString.Length == 8)
00259                     {
00260                         byte r = byte.Parse(cssString.Substring(0, 2), System.Globalization.NumberStyles.HexNumber);
00261                         byte g = byte.Parse(cssString.Substring(2, 2), System.Globalization.NumberStyles.HexNumber);
00262                         byte b = byte.Parse(cssString.Substring(4, 2), System.Globalization.NumberStyles.HexNumber);
00263                         byte a = byte.Parse(cssString.Substring(6, 2), System.Globalization.NumberStyles.HexNumber);
00264 
00265                         return Colour.FromRgba(r, g, b, a);
00266                     }
00267                     else
00268                     {
00269                         return null;
00270                     }
00271                 }
00272                 catch
00273                 {
00274                     return null;
00275                 }
```

```
00276             }
00277             else if (cssString.StartsWith("rgb(") || cssString.StartsWith("rgba("))
00278             {
00279                 try
00280                 {
00281                     cssString = cssString.Substring(cssString.IndexOf("(") + 1).Replace(")",
    "").Replace(" ", "");
00282                     string[] splitCssString = cssString.Split(',');
00283 
00284                     double R = ParseColourValueOrPercentage(splitCssString[0]);
00285                     double G = ParseColourValueOrPercentage(splitCssString[1]);
00286                     double B = ParseColourValueOrPercentage(splitCssString[2]);
00287 
00288                     double A = 1;
00289 
00290                     if (splitCssString.Length == 4)
00291                     {
00292                         A = double.Parse(splitCssString[3],
    System.Globalization.CultureInfo.InvariantCulture);
00293                     }
00294 
00295                     return Colour.FromRgba(R, G, B, A);
00296                 }
00297                 catch
00298                 {
00299                     return null;
00300                 }
00301             }
00302             else
00303             {
00304                 if (StandardColours.TryGetValue(cssString, out Colour tbr))
00305                 {
00306                     return tbr;
00307                 }
00308                 else
00309                 {
00310                     return null;
00311                 }
00312             }
00313         }
00314 
00315         private static double ParseColourValueOrPercentage(string value)
00316         {
00317             if (int.TryParse(value, out int tbr))
00318             {
00319                 return tbr / 255.0;
00320             }
00321             else if (value.Contains("%"))
00322             {
00323                 return double.Parse(value.Replace("%", ""),
    System.Globalization.CultureInfo.InvariantCulture) / 100.0;
00324             }
00325             else
00326             {
00327                 return double.Parse(value, System.Globalization.CultureInfo.InvariantCulture);
00328             }
00329         }
00330 
00331 /// <summary>
00332 /// Create a new <see cref="Colour"/> with the same RGB components as the <paramref name="original"/>
    <see cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00333 /// </summary>
00334 /// <param name="original">The original <see cref="Colour"/> from which the RGB components will be
    taken.</param>
00335 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00336 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the <paramref
    name="original"/> <see cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00337         public static Colour WithAlpha(Colour original, double alpha)
00338         {
00339             return Colour.FromRgba(original.R, original.G, original.B, alpha);
00340         }
00341 
00342 /// <summary>
00343 /// Create a new <see cref="Colour"/> with the same RGB components as the <paramref name="original"/>
    <see cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00344 /// </summary>
00345 /// <param name="original">The original <see cref="Colour"/> from which the RGB components will be
    taken.</param>
00346 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00347 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the <paramref
    name="original"/> <see cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00348         public static Colour WithAlpha(Colour original, byte alpha)
00349         {
00350             return Colour.FromRgba(original.R, original.G, original.B, (double)alpha / 255.0);
00351         }
00352 
00353 /// <summary>
```

```
00354 /// Create a new <see cref="Colour"/> with the same RGB components as the current <see
      cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00355 /// </summary>
00356 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00357 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the current <see
      cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00358         public Colour WithAlpha(double alpha)
00359         {
00360             return Colour.FromRgba(this.R, this.G, this.B, alpha);
00361         }
00362 
00363 /// <summary>
00364 /// Create a new <see cref="Colour"/> with the same RGB components as the current <see
      cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00365 /// </summary>
00366 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00367 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the current <see
      cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00368         public Colour WithAlpha(byte alpha)
00369         {
00370             return Colour.FromRgba(this.R, this.G, this.B, (double)alpha / 255.0);
00371         }
00372 
00373 /// <summary>
00374 /// Converts a <see cref="Colour"/> to the CIE XYZ colour space.
00375 /// </summary>
00376 /// <returns>A <see cref="ValueTuple"/> containing the X, Y and Z components of the <see
      cref="Colour"/>.</returns>
00377         public (double X, double Y, double Z) ToXYZ()
00378         {
00379             double r, g, b;
00380 
00381             if (R <= 0.04045)
00382             {
00383                 r = 25 * R / 323;
00384             }
00385             else
00386             {
00387                 r = Math.Pow((200 * R + 11) / 211, 2.4);
00388             }
00389 
00390             if (G <= 0.04045)
00391             {
00392                 g = 25 * G / 323;
00393             }
00394             else
00395             {
00396                 g = Math.Pow((200 * G + 11) / 211, 2.4);
00397             }
00398 
00399             if (B <= 0.04045)
00400             {
00401                 b = 25 * B / 323;
00402             }
00403             else
00404             {
00405                 b = Math.Pow((200 * B + 11) / 211, 2.4);
00406             }
00407 
00408             double x = 0.41239080 * r + 0.35758434 * g + 0.18048079 * b;
00409             double y = 0.21263901 * r + 0.71516868 * g + 0.07219232 * b;
00410             double z = 0.01933082 * r + 0.11919478 * g + 0.95053215 * b;
00411 
00412             return (x, y, z);
00413         }
00414 
00415 /// <summary>
00416 /// Creates a <see cref="Colour"/> from CIE XYZ coordinates.
00417 /// </summary>
00418 /// <param name="x">The X coordinate.</param>
00419 /// <param name="y">The Y coordinate.</param>
00420 /// <param name="z">The Z coordinate.</param>
00421 /// <returns>An sRGB <see cref="Colour"/> created from the specified components.</returns>
00422         public static Colour FromXYZ(double x, double y, double z)
00423         {
00424             double r = +3.24096994 * x - 1.53738318 * y - 0.49861076 * z;
00425             double g = -0.96924364 * x + 1.8759675 * y + 0.04155506 * z;
00426             double b = 0.05563008 * x - 0.20397696 * y + 1.05697151 * z;
00427 
00428             if (r <= 0.0031308)
00429             {
00430                 r = 323 * r / 25;
00431             }
00432             else
00433             {
00434                 r = (211 * Math.Pow(r, 1 / 2.4) - 11) / 200;
00435             }
```

```
00436
00437             if (g <= 0.0031308)
00438             {
00439                 g = 323 * g / 25;
00440             }
00441             else
00442             {
00443                 g = (211 * Math.Pow(g, 1 / 2.4) - 11) / 200;
00444             }
00445
00446             if (b <= 0.0031308)
00447             {
00448                 b = 323 * b / 25;
00449             }
00450             else
00451             {
00452                 b = (211 * Math.Pow(b, 1 / 2.4) - 11) / 200;
00453             }
00454
00455             r = Math.Min(Math.Max(0, r), 1);
00456             g = Math.Min(Math.Max(0, g), 1);
00457             b = Math.Min(Math.Max(0, b), 1);
00458
00459             return Colour.FromRgb(r, g, b);
00460         }
00461
00462 /// <summary>
00463 /// Converts a <see cref="Colour"/> to the CIE Lab colour space (under Illuminant D65).
00464 /// </summary>
00465 /// <returns>A <see cref="ValueType"/> containing the L*, a* and b* components of the <see
      cref="Colour"/>.</returns>
00466         public (double L, double a, double b) ToLab()
00467         {
00468             double f(double t)
00469             {
00470                 const double d = 6.0 / 29;
00471
00472                 if (t > d * d * d)
00473                 {
00474                     return Math.Pow(t, 1.0 / 3);
00475                 }
00476                 else
00477                 {
00478                     return t / (3 * d * d) + 4.0 / 29;
00479                 }
00480             }
00481
00482             const double xN = 0.950489;
00483             const double yN = 1;
00484             const double zN = 1.088840;
00485
00486             (double x, double y, double z) = this.ToXYZ();
00487
00488             double fY = f(y / yN);
00489
00490             double l = 1.16 * fY - 0.16;
00491             double a = 5 * (f(x / xN) - fY);
00492             double b = 2 * (fY - f(z / zN));
00493
00494             return (l, a, b);
00495         }
00496
00497 /// <summary>
00498 /// Creates a <see cref="Colour"/> from CIE Lab coordinates (under Illuminant D65).
00499 /// </summary>
00500 /// <param name="L">The L* component.</param>
00501 /// <param name="a">The a* component.</param>
00502 /// <param name="b">The b* component.</param>
00503 /// <returns>An sRGB <see cref="Colour"/> created from the specified components.</returns>
00504         public static Colour FromLab(double L, double a, double b)
00505         {
00506             double f(double t)
00507             {
00508                 const double d = 6.0 / 29;
00509
00510                 if (t > d)
00511                 {
00512                     return t * t * t;
00513                 }
00514                 else
00515                 {
00516                     return 3 * d * d * (t - 4.0 / 29);
00517                 }
00518             }
00519
00520             const double xN = 0.950489;
00521             const double yN = 1;
```

```
00522             const double zN = 1.088840;
00523
00524             double x = xN * f((L + 0.16) / 1.16 + a / 5);
00525             double y = yN * f((L + 0.16) / 1.16);
00526             double z = zN * f((L + 0.16) / 1.16 - b / 2);
00527
00528             return Colour.FromXYZ(x, y, z);
00529         }
00530
00531 /// <summary>
00532 /// Converts a <see cref="Colour"/> to the HSL colour space.
00533 /// </summary>
00534 /// <returns>A <see cref="ValueType"/> containing the H, S and L components of the <see
      cref="Colour"/>.  Each component has range [0, 1].</returns>
00535         public (double H, double S, double L) ToHSL()
00536         {
00537             double xMax = Math.Max(Math.Max(R, G), B);
00538             double xMin = Math.Min(Math.Min(R, G), B);
00539
00540             double l = (xMax + xMin) * 0.5;
00541
00542             double h;
00543
00544             if (xMax == xMin)
00545             {
00546                 h = 0;
00547             }
00548             else if (xMax == R)
00549             {
00550                 h = (G - B) / (xMax - xMin) / 6;
00551             }
00552             else if (xMax == G)
00553             {
00554                 h = (2 + (B - R) / (xMax - xMin)) / 6;
00555             }
00556             else
00557             {
00558                 h = (4 + (R - G) / (xMax - xMin)) / 6;
00559             }
00560
00561             double s;
00562
00563             if (l == 0 || l == 1)
00564             {
00565                 s = 0;
00566             }
00567             else
00568             {
00569                 s = (xMax - l) / Math.Min(l, 1 - l);
00570             }
00571
00572             return (h, s, l);
00573         }
00574
00575 /// <summary>
00576 /// Creates a <see cref="Colour"/> from HSL coordinates.
00577 /// </summary>
00578 /// <param name="h">The H component.  Should be in range [0, 1].</param>
00579 /// <param name="s">The S component.  Should be in range [0, 1].</param>
00580 /// <param name="l">The L component.  Should be in range [0, 1].</param>
00581 /// <returns>A <see cref="Colour"/> created from the specified components.</returns>
00582         public static Colour FromHSL(double h, double s, double l)
00583         {
00584             double c = (1 - Math.Abs(2 * l - 1)) * s;
00585
00586             double hp = h * 6;
00587
00588             double x = c * (1 - Math.Abs((hp % 2) - 1));
00589
00590             double r1, g1, b1;
00591
00592             if (hp <= 1)
00593             {
00594                 r1 = c;
00595                 g1 = x;
00596                 b1 = 0;
00597             }
00598             else if (hp <= 2)
00599             {
00600                 r1 = x;
00601                 g1 = c;
00602                 b1 = 0;
00603             }
00604             else if (hp <= 3)
00605             {
00606                 r1 = 0;
00607                 g1 = c;
```

```
00608                 b1 = x;
00609             }
00610             else if (hp <= 4)
00611             {
00612                 r1 = 0;
00613                 g1 = x;
00614                 b1 = c;
00615             }
00616             else if (hp <= 5)
00617             {
00618                 r1 = x;
00619                 g1 = 0;
00620                 b1 = c;
00621             }
00622             else if (hp <= 6)
00623             {
00624                 r1 = c;
00625                 g1 = 0;
00626                 b1 = x;
00627             }
00628             else
00629             {
00630                 r1 = 0;
00631                 g1 = 0;
00632                 b1 = 0;
00633             }
00634 
00635             double m = l - c / 2;
00636 
00637             return Colour.FromRgb(r1 + m, g1 + m, b1 + m);
00638         }
00639 
00640     }
00641 }
```

8.53 Document.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021 
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents a collection of pages.
00026 /// </summary>
00027     public class Document
00028     {
00029 /// <summary>
00030 /// The pages in the document.
00031 /// </summary>
00032         public List<Page> Pages = new List<Page>();
00033 
00034 
00035 /// <summary>
00036 /// Create a new document.
00037 /// </summary>
00038         public Document()
00039         {
00040 
00041         }
00042     }
00043 
00044 /// <summary>
00045 /// Represents a <see cref="Graphics"/> object with a width and height.
00046 /// </summary>
```

```
00047     public class Page
00048     {
00049 /// <summary>
00050 /// Width of the page.
00051 /// </summary>
00052         public double Width { get; set; }
00053
00054 /// <summary>
00055 /// Height of the page.
00056 /// </summary>
00057         public double Height { get; set; }
00058
00059 /// <summary>
00060 /// Graphics surface of the page.
00061 /// </summary>
00062         public Graphics Graphics { get; set; }
00063
00064 /// <summary>
00065 /// Background colour of the page.
00066 /// </summary>
00067         public Colour Background { get; set; } = Colour.FromRgba(255, 255, 255, 0);
00068
00069 /// <summary>
00070 /// Create a new page.
00071 /// </summary>
00072 /// <param name="width">The width of the page.</param>
00073 /// <param name="height">The height of the page.</param>
00074         public Page(double width, double height)
00075         {
00076             this.Width = width;
00077             this.Height = height;
00078
00079             this.Graphics = new Graphics();
00080             this.Graphics.Translate(0, 0);
00081         }
00082
00083 /// <summary>
00084 /// Translate and resize the <see cref="Page"/> so that it displays the specified <paramref
      name="region"/>.
00085 /// </summary>
00086 /// <param name="region">The area to include in the page.</param>
00087 /// <param name="removeClippedGraphics">If this is <see langword="true"/>, graphics actions that fall
      outside of the specified region are completely removed from the plot, otherwise they are just
      hidden.</param>
00088 /// <param name="tag">A tag to identify the transform.</param>
00089         public void Crop(Rectangle region, bool removeClippedGraphics = false, string tag = null)
00090         {
00091             this.Crop(region.Location, region.Size, removeClippedGraphics, tag);
00092         }
00093
00094 /// <summary>
00095 /// Translate and resize the <see cref="Page"/> so that it displays the rectangle defined by <paramref
      name="topLeft"/> and <paramref name="size"/>.
00096 /// </summary>
00097 /// <param name="topLeft">The top left corner of the area to include in the page.</param>
00098 /// <param name="size">The size of the area to include in the page.</param>
00099 /// <param name="removeClippedGraphics">If this is <see langword="true"/>, graphics actions that fall
      outside of the specified region are completely removed from the plot, otherwise they are just
      hidden.</param>
00100 /// <param name="tag">A tag to identify the transform.</param>
00101         public void Crop(Point topLeft, Size size, bool removeClippedGraphics = false, string tag =
      null)
00102         {
00103             if (removeClippedGraphics)
00104             {
00105                 this.Graphics.Crop(new Rectangle(topLeft, size));
00106             }
00107
00108             if (this.Graphics.Actions[0] is TransformAction transf)
00109             {
00110                 double[,] currMatrix = transf.GetMatrix();
00111
00112                 double[,] newMatrix = Graphics.Multiply(Graphics.TranslationMatrix(-topLeft.X,
      -topLeft.Y), currMatrix);
00113
00114                 this.Graphics.Actions[0] = new TransformAction(newMatrix, tag);
00115             }
00116             else
00117             {
00118                 this.Graphics.Actions.Insert(0, new TransformAction(new Point(-topLeft.X, -topLeft.Y),
      tag));
00119             }
00120
00121             this.Width = size.Width;
00122             this.Height = size.Height;
00123         }
00124
```

```
00125 /// <summary>
00126 /// Translate and resize the <see cref="Page"/> so that it displays the rectangle corresponding to the
      bounding box of its contents.
00127 /// </summary>
00128 /// <param name="tag">A tag to identify the transform.</param>
00129         public void Crop(string tag = null)
00130         {
00131             Rectangle bounds = this.Graphics.GetBounds();
00132 
00133             if (this.Graphics.Actions[0] is TransformAction transf)
00134             {
00135                 double[,] currMatrix = transf.GetMatrix();
00136 
00137                 double[,] newMatrix = Graphics.Multiply(Graphics.TranslationMatrix(-bounds.Location.X,
      -bounds.Location.Y), currMatrix);
00138 
00139                 this.Graphics.Actions[0] = new TransformAction(newMatrix, tag);
00140             }
00141             else
00142             {
00143                 this.Graphics.Actions.Insert(0, new TransformAction(new Point(-bounds.Location.X,
      -bounds.Location.Y), tag));
00144             }
00145 
00146             this.Width = bounds.Size.Width;
00147             this.Height = bounds.Size.Height;
00148         }
00149     }
00150 }
```

8.54 Enums.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 namespace VectSharp
00019 {
00020 /// <summary>
00021 /// Represent text baselines.
00022 /// </summary>
00023     public enum TextBaselines
00024     {
00025 /// <summary>
00026 /// The current vertical coordinate determines where the top of the text string will be placed.
00027 /// </summary>
00028         Top,
00029 
00030 /// <summary>
00031 /// The current vertical coordinate determines where the bottom of the text string will be placed.
00032 /// </summary>
00033         Bottom,
00034 
00035 /// <summary>
00036 /// The current vertical coordinate determines where the middle of the text string will be placed.
00037 /// </summary>
00038         Middle,
00039 
00040 /// <summary>
00041 /// The current vertical coordinate determines where the baseline of the text string will be placed.
00042 /// </summary>
00043         Baseline
00044     }
00045 
00046 /// <summary>
00047 /// Represents text anchors.
00048 /// </summary>
00049     public enum TextAnchors
00050     {
00051 /// <summary>
```

```
00052 /// The current coordinate will determine the position of the left side of the text string.
00053 /// </summary>
00054         Left,
00055
00056 /// <summary>
00057 /// The current coordinate will determine the position of the center of the text string.
00058 /// </summary>
00059         Center,
00060
00061 /// <summary>
00062 /// The current coordinate will determine the position of the right side of the text string.
00063 /// </summary>
00064         Right
00065     }
00066
00067 /// <summary>
00068 /// Represents line caps.
00069 /// </summary>
00070     public enum LineCaps
00071     {
00072 /// <summary>
00073 /// The ends of the line are squared off at the endpoints.
00074 /// </summary>
00075         Butt = 0,
00076
00077 /// <summary>
00078 /// The ends of the lines are rounded.
00079 /// </summary>
00080         Round = 1,
00081
00082 /// <summary>
00083 /// The ends of the lines are squared off by adding an half square box at each end.
00084 /// </summary>
00085         Square = 2
00086     }
00087
00088 /// <summary>
00089 /// Represents line joining options.
00090 /// </summary>
00091     public enum LineJoins
00092     {
00093 /// <summary>
00094 /// Consecutive segments are joined by straight corners.
00095 /// </summary>
00096         Bevel = 2,
00097
00098 /// <summary>
00099 /// Consecutive segments are joined by extending their outside edges until they meet.
00100 /// </summary>
00101         Miter = 0,
00102
00103 /// <summary>
00104 /// Consecutive segments are joined by arc segments.
00105 /// </summary>
00106         Round = 1
00107     }
00108
00109 /// <summary>
00110 /// Represents instructions on how to paint a dashed line.
00111 /// </summary>
00112     public struct LineDash
00113     {
00114 /// <summary>
00115 /// A solid (not dashed) line
00116 /// </summary>
00117         public static LineDash SolidLine = new LineDash(0, 0, 0);
00118
00119 /// <summary>
00120 /// Length of the "on" (painted) segment.
00121 /// </summary>
00122         public double UnitsOn;
00123
00124 /// <summary>
00125 /// Length of the "off" (not painted) segment.
00126 /// </summary>
00127         public double UnitsOff;
00128
00129 /// <summary>
00130 /// Position in the dash pattern at which the line starts.
00131 /// </summary>
00132         public double Phase;
00133
00134 /// <summary>
00135 /// Define a new line dash pattern.
00136 /// </summary>
00137 /// <param name="unitsOn">The length of the "on" (painted) segment.</param>
00138 /// <param name="unitsOff">The length of the "off" (not painted) segment.</param>
```

```
00139 /// <param name="phase">The position in the dash pattern at which the line starts.</param>
00140         public LineDash(double unitsOn, double unitsOff, double phase)
00141         {
00142             UnitsOn = unitsOn;
00143             UnitsOff = unitsOff;
00144             Phase = phase;
00145         }
00146     }
00147 
00148 /// <summary>
00149 /// Types of <see cref="Segment"/>.
00150 /// </summary>
00151     public enum SegmentType
00152     {
00153 /// <summary>
00154 /// The segment represents a move from the current point to a new point.
00155 /// </summary>
00156         Move,
00157 
00158 /// <summary>
00159 /// The segment represents a straight line from the current point to a new point.
00160 /// </summary>
00161         Line,
00162 
00163 /// <summary>
00164 /// The segment represents a cubic bezier curve from the current point to a new point.
00165 /// </summary>
00166         CubicBezier,
00167 
00168 /// <summary>
00169 /// The segment represents a circular arc from the current point to a new point.
00170 /// </summary>
00171         Arc,
00172 
00173 /// <summary>
00174 /// The segment represents the closing segment of a figure.
00175 /// </summary>
00176         Close
00177     }
00178 
00179 /// <summary>
00180 /// Represents ways to deal with unbalanced graphics state stacks.
00181 /// </summary>
00182     public enum UnbalancedStackActions
00183     {
00184 /// <summary>
00185 /// If the graphics state stack is unbalanced, an exception will be thrown.
00186 /// </summary>
00187         Throw,
00188 
00189 /// <summary>
00190 /// The graphics state stack will be automatically balanced by adding or removing calls to <see
      cref="Graphics.Restore"/> as necessary.
00191 /// </summary>
00192         SilentlyFix,
00193 
00194 /// <summary>
00195 /// No attempt will be made at correcting an unbalanced graphics state stack.  This may cause issues
      with some consumers.
00196 /// </summary>
00197         Ignore
00198     }
00199 }
```

8.55 BoxBlurFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
```

```
00018 using System;
00019 using System.Threading.Tasks;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a filter applying a box blur.
00025 /// </summary>
00026     public class BoxBlurFilter :  ILocationInvariantFilter
00027     {
00028 /// <summary>
00029 /// The radius of the box blur (the actual size of the box is 2 * <see cref="BoxRadius"/> + 1).
00030 /// </summary>
00031         public double BoxRadius { get; }
00032
00033 /// <inheritdoc/>
00034         public Point TopLeftMargin { get; }
00035 /// <inheritdoc/>
00036         public Point BottomRightMargin { get; }
00037
00038 /// <summary>
00039 /// Creates a new <see cref="BoxBlurFilter"/> with the specified radius.
00040 /// </summary>
00041 /// <param name="boxRadius">The radius of the box blur (the actual size of the box is 2 * <paramref
      name="boxRadius"/> + 1).</param>
00042         public BoxBlurFilter(double boxRadius)
00043         {
00044             this.BoxRadius = boxRadius;
00045             this.TopLeftMargin = new Point(boxRadius, boxRadius);
00046             this.BottomRightMargin = new Point(boxRadius, boxRadius);
00047         }
00048
00049 /// <inheritdoc/>
00050         public RasterImage Filter(RasterImage image, double scale)
00051         {
00052             return BoxBlurSRGB(image, (int)Math.Round(this.BoxRadius * scale));
00053         }
00054
00055         private RasterImage BoxBlurSRGB(RasterImage image, int boxRadius)
00056         {
00057             IntPtr intermediateData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width
      * image.Height * (image.HasAlpha ?  4 :  3));
00058             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00059             GC.AddMemoryPressure(2 * image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00060
00061             int width = image.Width;
00062             int height = image.Height;
00063
00064             int pixelSize = image.HasAlpha ?  4 :  3;
00065             int stride = image.Width * pixelSize;
00066
00067             double normFactor = 1.0 / (2 * boxRadius + 1);
00068
00069             int threads;
00070
00071             double size = Math.Sqrt((double)image.Width * image.Height);
00072
00073             if (size <= 128)
00074             {
00075                 threads = 1;
00076             }
00077             else if (size <= 512)
00078             {
00079                 threads = Math.Min(4, Environment.ProcessorCount);
00080             }
00081             else
00082             {
00083                 threads = Math.Min(8, Environment.ProcessorCount);
00084             }
00085
00086             unsafe
00087             {
00088                 byte* input = (byte*)image.ImageDataAddress;
00089                 byte* intermediate = (byte*)intermediateData;
00090                 byte* output = (byte*)tbrData;
00091
00092                 Action<int> yLoop;
00093
00094                 if (image.HasAlpha)
00095                 {
00096                     yLoop = y =>
00097                     {
00098                         double rAcc = 0;
00099                         double gAcc = 0;
00100                         double bAcc = 0;
00101                         double aAcc = 0;
```

```
00102
00103                           double prevR = 0;
00104                           double prevG = 0;
00105                           double prevB = 0;
00106                           double prevA = 0;
00107
00108                           double r = 0;
00109                           double g = 0;
00110                           double b = 0;
00111
00112                           double a = input[y * stride + 3] / 255.0;
00113
00114                           prevR = input[y * stride] * a;
00115                           prevG = input[y * stride + 1] * a;
00116                           prevB = input[y * stride + 2] * a;
00117                           prevA = a;
00118
00119                           rAcc = prevR * (boxRadius + 1);
00120                           gAcc = prevG * (boxRadius + 1);
00121                           bAcc = prevB * (boxRadius + 1);
00122                           aAcc = a * (boxRadius + 1);
00123
00124                           int rX = 0;
00125
00126                           for (int x = 0; x < boxRadius; x++)
00127                           {
00128                               rX = Math.Min(x, width - 1);
00129
00130                               a = input[y * stride + rX * 4 + 3] / 255.0;
00131
00132                               rAcc += input[y * stride + rX * 4] * a;
00133                               gAcc += input[y * stride + rX * 4 + 1] * a;
00134                               bAcc += input[y * stride + rX * 4 + 2] * a;
00135                               aAcc += a;
00136                           }
00137
00138                           rX = Math.Min(boxRadius, width - 1);
00139                           int lX = 0;
00140
00141                           for (int x = 0; x < width; x++)
00142                           {
00143                               rX = Math.Min(x + boxRadius, width - 1);
00144                               lX = Math.Max(x - boxRadius - 1, 0);
00145
00146                               a = input[y * stride + rX * 4 + 3] / 255.0;
00147                               r = input[y * stride + rX * 4] * a;
00148                               g = input[y * stride + rX * 4 + 1] * a;
00149                               b = input[y * stride + rX * 4 + 2] * a;
00150
00151                               prevA = input[y * stride + lX * 4 + 3] / 255.0;
00152                               prevR = input[y * stride + lX * 4] * prevA;
00153                               prevG = input[y * stride + lX * 4 + 1] * prevA;
00154                               prevB = input[y * stride + lX * 4 + 2] * prevA;
00155
00156                               rAcc += r - prevR;
00157                               gAcc += g - prevG;
00158                               bAcc += b - prevB;
00159                               aAcc += a - prevA;
00160
00161                               if (aAcc != 0)
00162                               {
00163                                   intermediate[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0,
      rAcc / aAcc));
00164                                   intermediate[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0,
      gAcc / aAcc));
00165                                   intermediate[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0,
      bAcc / aAcc));
00166                                   intermediate[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      aAcc * 255 * normFactor));
00167                               }
00168                               else
00169                               {
00170                                   intermediate[y * stride + x * 4] = 0;
00171                                   intermediate[y * stride + x * 4 + 1] = 0;
00172                                   intermediate[y * stride + x * 4 + 2] = 0;
00173                                   intermediate[y * stride + x * 4 + 3] = 0;
00174                               }
00175                           }
00176                       };
00177                   }
00178                   else
00179                   {
00180                       yLoop = y =>
00181                       {
00182                           double rAcc = 0;
00183                           double gAcc = 0;
00184                           double bAcc = 0;
```

```
00185
00186                       double prevR = 0;
00187                       double prevG = 0;
00188                       double prevB = 0;
00189
00190                       double r = 0;
00191                       double g = 0;
00192                       double b = 0;
00193
00194                       prevR = input[y * stride];
00195                       prevG = input[y * stride + 1];
00196                       prevB = input[y * stride + 2];
00197
00198                       rAcc = prevR * (boxRadius + 1);
00199                       gAcc = prevG * (boxRadius + 1);
00200                       bAcc = prevB * (boxRadius + 1);
00201
00202                       int rX = 0;
00203
00204                       for (int x = 0; x < boxRadius; x++)
00205                       {
00206                           rX = Math.Min(x, width - 1);
00207
00208                           rAcc += input[y * stride + rX * 3];
00209                           gAcc += input[y * stride + rX * 3 + 1];
00210                           bAcc += input[y * stride + rX * 3 + 2];
00211                       }
00212
00213                       rX = Math.Min(boxRadius, width - 1);
00214                       int lX = 0;
00215
00216                       for (int x = 0; x < width; x++)
00217                       {
00218                           rX = Math.Min(x + boxRadius, width - 1);
00219                           lX = Math.Max(x - boxRadius - 1, 0);
00220
00221                           r = input[y * stride + rX * 3];
00222                           g = input[y * stride + rX * 3 + 1];
00223                           b = input[y * stride + rX * 3 + 2];
00224
00225                           prevR = input[y * stride + lX * 3];
00226                           prevG = input[y * stride + lX * 3 + 1];
00227                           prevB = input[y * stride + lX * 3 + 2];
00228
00229                           rAcc += r - prevR;
00230                           gAcc += g - prevG;
00231                           bAcc += b - prevB;
00232
00233                           intermediate[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, rAcc *
      normFactor));
00234                           intermediate[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0,
      gAcc * normFactor));
00235                           intermediate[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0,
      bAcc * normFactor));
00236                       }
00237                   };
00238               }
00239
00240               Action<int> xLoop;
00241
00242               if (image.HasAlpha)
00243               {
00244                   xLoop = x =>
00245                   {
00246                       double rAcc = 0;
00247                       double gAcc = 0;
00248                       double bAcc = 0;
00249                       double aAcc = 0;
00250
00251                       double prevR = 0;
00252                       double prevG = 0;
00253                       double prevB = 0;
00254                       double prevA = 0;
00255
00256                       double r = 0;
00257                       double g = 0;
00258                       double b = 0;
00259
00260                       double a = intermediate[x * 4 + 3] / 255.0;
00261
00262                       prevR = intermediate[x * 4] * a;
00263                       prevG = intermediate[x * 4 + 1] * a;
00264                       prevB = intermediate[x * 4 + 2] * a;
00265                       prevA = a;
00266
00267                       rAcc = prevR * (boxRadius + 1);
00268                       gAcc = prevG * (boxRadius + 1);
```

```
00269                         bAcc = prevB * (boxRadius + 1);
00270                         aAcc = a * (boxRadius + 1);
00271
00272                         int rY = 0;
00273
00274                         for (int y = 0; y < boxRadius; y++)
00275                         {
00276                             rY = Math.Min(y, height - 1);
00277
00278                             a = intermediate[rY * stride + x * 4 + 3] / 255.0;
00279
00280                             rAcc += intermediate[rY * stride + x * 4] * a;
00281                             gAcc += intermediate[rY * stride + x * 4 + 1] * a;
00282                             bAcc += intermediate[rY * stride + x * 4 + 2] * a;
00283                             aAcc += a;
00284                         }
00285
00286                         rY = Math.Min(boxRadius, height - 1);
00287                         int lY = 0;
00288
00289                         for (int y = 0; y < height; y++)
00290                         {
00291                             rY = Math.Min(y + boxRadius, height - 1);
00292                             lY = Math.Max(y - boxRadius - 1, 0);
00293
00294                             a = intermediate[rY * stride + x * 4 + 3] / 255.0;
00295                             r = intermediate[rY * stride + x * 4] * a;
00296                             g = intermediate[rY * stride + x * 4 + 1] * a;
00297                             b = intermediate[rY * stride + x * 4 + 2] * a;
00298
00299                             prevA = intermediate[lY * stride + x * 4 + 3] / 255.0;
00300                             prevR = intermediate[lY * stride + x * 4] * prevA;
00301                             prevG = intermediate[lY * stride + x * 4 + 1] * prevA;
00302                             prevB = intermediate[lY * stride + x * 4 + 2] * prevA;
00303
00304                             rAcc += r - prevR;
00305                             gAcc += g - prevG;
00306                             bAcc += b - prevB;
00307                             aAcc += a - prevA;
00308
00309                             if (aAcc != 0)
00310                             {
00311                                 output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, rAcc /
      aAcc));
00312                                 output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, gAcc
      / aAcc));
00313                                 output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, bAcc
      / aAcc));
00314                                 output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0, aAcc
      * 255 * normFactor));
00315                             }
00316                             else
00317                             {
00318                                 output[y * stride + x * 4] = 0;
00319                                 output[y * stride + x * 4 + 1] = 0;
00320                                 output[y * stride + x * 4 + 2] = 0;
00321                                 output[y * stride + x * 4 + 3] = 0;
00322                             }
00323                         }
00324                     };
00325                 }
00326                 else
00327                 {
00328                     xLoop = x =>
00329                     {
00330                         double rAcc = 0;
00331                         double gAcc = 0;
00332                         double bAcc = 0;
00333
00334                         double prevR = 0;
00335                         double prevG = 0;
00336                         double prevB = 0;
00337
00338                         double r = 0;
00339                         double g = 0;
00340                         double b = 0;
00341
00342                         prevR = intermediate[x * 3];
00343                         prevG = intermediate[x * 3 + 1];
00344                         prevB = intermediate[x * 3 + 2];
00345
00346                         rAcc = prevR * (boxRadius + 1);
00347                         gAcc = prevG * (boxRadius + 1);
00348                         bAcc = prevB * (boxRadius + 1);
00349
00350                         int rY = 0;
00351
```

```
00352                         for (int y = 0; y < boxRadius; y++)
00353                         {
00354                             rY = Math.Min(y, height - 1);
00355 
00356                             rAcc += intermediate[rY * stride + x * 3];
00357                             gAcc += intermediate[rY * stride + x * 3 + 1];
00358                             bAcc += intermediate[rY * stride + x * 3 + 2];
00359                         }
00360 
00361                         rY = Math.Min(boxRadius, height - 1);
00362                         int lY = 0;
00363 
00364                         for (int y = 0; y < height; y++)
00365                         {
00366                             rY = Math.Min(y + boxRadius, height - 1);
00367                             lY = Math.Max(y - boxRadius - 1, 0);
00368 
00369                             r = intermediate[rY * stride + x * 3];
00370                             g = intermediate[rY * stride + x * 3 + 1];
00371                             b = intermediate[rY * stride + x * 3 + 2];
00372 
00373                             prevR = intermediate[lY * stride + x * 3];
00374                             prevG = intermediate[lY * stride + x * 3 + 1];
00375                             prevB = intermediate[lY * stride + x * 3 + 2];
00376 
00377                             rAcc += r - prevR;
00378                             gAcc += g - prevG;
00379                             bAcc += b - prevB;
00380 
00381                             output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, rAcc *
      normFactor));
00382                             output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, gAcc *
      normFactor));
00383                             output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, bAcc *
      normFactor));
00384                         }
00385                     };
00386                 }
00387 
00388                 if (threads == 1)
00389                 {
00390                     for (int y = 0; y < height; y++)
00391                     {
00392                         yLoop(y);
00393                     }
00394 
00395                     for (int x = 0; x < width; x++)
00396                     {
00397                         xLoop(x);
00398                     }
00399                 }
00400                 else
00401                 {
00402                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00403 
00404                     Parallel.For(0, height, options, yLoop);
00405                     Parallel.For(0, width, options, xLoop);
00406                 }
00407             }
00408 
00409             System.Runtime.InteropServices.Marshal.FreeHGlobal(intermediateData);
00410             GC.RemoveMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00411 
00412             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00413 
00414             return new RasterImage(ref disp, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
00415         }
00416     }
00417 }
```

8.56 ColourMatrixFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
```

```
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Threading.Tasks;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a colour transformation matrix.
00025 /// </summary>
00026     public class ColourMatrix
00027     {
00028 /// <summary>
00029 /// The coefficient relating the R component of the output colour to the R component of the input
      colour.
00030 /// </summary>
00031         public double R1 { get; set; }
00032
00033 /// <summary>
00034 /// The coefficient relating the R component of the output colour to the G component of the input
      colour.
00035 /// </summary>
00036         public double R2 { get; set; }
00037
00038 /// <summary>
00039 /// The coefficient relating the R component of the output colour to the B component of the input
      colour.
00040 /// </summary>
00041         public double R3 { get; set; }
00042
00043 /// <summary>
00044 /// The coefficient relating the R component of the output colour to the A component of the input
      colour.
00045 /// </summary>
00046         public double R4 { get; set; }
00047
00048 /// <summary>
00049 /// The bias (translation) applied to the R component of the output colour.
00050 /// </summary>
00051         public double R5 { get; set; }
00052
00053 /// <summary>
00054 /// The coefficient relating the G component of the output colour to the R component of the input
      colour.
00055 /// </summary>
00056         public double G1 { get; set; }
00057
00058 /// <summary>
00059 /// The coefficient relating the G component of the output colour to the G component of the input
      colour.
00060 /// </summary>
00061         public double G2 { get; set; }
00062
00063 /// <summary>
00064 /// The coefficient relating the G component of the output colour to the B component of the input
      colour.
00065 /// </summary>
00066         public double G3 { get; set; }
00067
00068 /// <summary>
00069 /// The coefficient relating the G component of the output colour to the A component of the input
      colour.
00070 /// </summary>
00071         public double G4 { get; set; }
00072
00073 /// <summary>
00074 /// The bias (translation) applied to the R component of the output colour.
00075 /// </summary>
00076         public double G5 { get; set; }
00077
00078 /// <summary>
00079 /// The coefficient relating the B component of the output colour to the R component of the input
      colour.
00080 /// </summary>
00081         public double B1 { get; set; }
00082
00083 /// <summary>
00084 /// The coefficient relating the B component of the output colour to the G component of the input
      colour.
00085 /// </summary>
00086         public double B2 { get; set; }
```

```
00087
00088 /// <summary>
00089 /// The coefficient relating the B component of the output colour to the B component of the input
      colour.
00090 /// </summary>
00091         public double B3 { get; set; }
00092
00093 /// <summary>
00094 /// The coefficient relating the B component of the output colour to the A component of the input
      colour.
00095 /// </summary>
00096         public double B4 { get; set; }
00097
00098 /// <summary>
00099 /// The bias (translation) applied to the B component of the output colour.
00100 /// </summary>
00101         public double B5 { get; set; }
00102
00103 /// <summary>
00104 /// The coefficient relating the A component of the output colour to the R component of the input
      colour.
00105 /// </summary>
00106         public double A1 { get; set; }
00107
00108 /// <summary>
00109 /// The coefficient relating the A component of the output colour to the G component of the input
      colour.
00110 /// </summary>
00111         public double A2 { get; set; }
00112
00113 /// <summary>
00114 /// The coefficient relating the A component of the output colour to the B component of the input
      colour.
00115 /// </summary>
00116         public double A3 { get; set; }
00117
00118 /// <summary>
00119 /// The coefficient relating the A component of the output colour to the A component of the input
      colour.
00120 /// </summary>
00121         public double A4 { get; set; }
00122
00123 /// <summary>
00124 /// The bias (translation) applied to the A component of the output colour.
00125 /// </summary>
00126         public double A5 { get; set; }
00127
00128 /// <summary>
00129 /// Gets or sets the requested element of the matrix.  Elements of the last row of the matrix can be
      read, but not set.
00130 /// </summary>
00131 /// <param name="y">The row of the matrix.</param>
00132 /// <param name="x">The column of the matrix.</param>
00133 /// <returns>The requested element of the matrix.</returns>
00134 /// <exception cref="ArgumentOutOfRangeException">An attempt has been made to access an element out of
      the bounds of the matrix.</exception>
00135         public double this[int y, int x]
00136         {
00137             get
00138             {
00139                 switch (y)
00140                 {
00141                     case 0:
00142                         switch (x)
00143                         {
00144                             case 0:
00145                                 return R1;
00146                             case 1:
00147                                 return R2;
00148                             case 2:
00149                                 return R3;
00150                             case 3:
00151                                 return R4;
00152                             case 4:
00153                                 return R5;
00154                             default:
00155                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00156                         }
00157
00158                     case 1:
00159                         switch (x)
00160                         {
00161                             case 0:
00162                                 return G1;
00163                             case 1:
00164                                 return G2;
```

```
00165                               case 2:
00166                                   return G3;
00167                               case 3:
00168                                   return G4;
00169                               case 4:
00170                                   return G5;
00171                               default:
00172                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00173                           }
00174
00175                       case 2:
00176                           switch (x)
00177                           {
00178                               case 0:
00179                                   return B1;
00180                               case 1:
00181                                   return B2;
00182                               case 2:
00183                                   return B3;
00184                               case 3:
00185                                   return B4;
00186                               case 4:
00187                                   return B5;
00188                               default:
00189                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00190                           }
00191
00192                       case 3:
00193                           switch (x)
00194                           {
00195                               case 0:
00196                                   return A1;
00197                               case 1:
00198                                   return A2;
00199                               case 2:
00200                                   return A3;
00201                               case 3:
00202                                   return A4;
00203                               case 4:
00204                                   return A5;
00205                               default:
00206                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00207                           }
00208
00209                       case 4:
00210                           switch (x)
00211                           {
00212                               case 0:
00213                                   return 0;
00214                               case 1:
00215                                   return 0;
00216                               case 2:
00217                                   return 0;
00218                               case 3:
00219                                   return 0;
00220                               case 4:
00221                                   return 1;
00222                               default:
00223                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00224                           }
00225
00226                       default:
00227                           throw new ArgumentOutOfRangeException(nameof(y), y, "Coordinate out of
      range!");
00228                   }
00229               }
00230
00231               private set
00232               {
00233                   switch (y)
00234                   {
00235                       case 0:
00236                           switch (x)
00237                           {
00238                               case 0:
00239                                   R1 = value;
00240                                   break;
00241                               case 1:
00242                                   R2 = value;
00243                                   break;
00244                               case 2:
00245                                   R3 = value;
00246                                   break;
```

```
00247                             case 3:
00248                                 R4 = value;
00249                                 break;
00250                             case 4:
00251                                 R5 = value;
00252                                 break;
00253                             default:
00254                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00255                         }
00256                         break;
00257
00258                     case 1:
00259                         switch (x)
00260                         {
00261                             case 0:
00262                                 G1 = value;
00263                                 break;
00264                             case 1:
00265                                 G2 = value;
00266                                 break;
00267                             case 2:
00268                                 G3 = value;
00269                                 break;
00270                             case 3:
00271                                 G4 = value;
00272                                 break;
00273                             case 4:
00274                                 G5 = value;
00275                                 break;
00276                             default:
00277                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00278                         }
00279                         break;
00280
00281                     case 2:
00282                         switch (x)
00283                         {
00284                             case 0:
00285                                 B1 = value;
00286                                 break;
00287                             case 1:
00288                                 B2 = value;
00289                                 break;
00290                             case 2:
00291                                 B3 = value;
00292                                 break;
00293                             case 3:
00294                                 B4 = value;
00295                                 break;
00296                             case 4:
00297                                 B5 = value;
00298                                 break;
00299                             default:
00300                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00301                         }
00302                         break;
00303
00304                     case 3:
00305                         switch (x)
00306                         {
00307                             case 0:
00308                                 A1 = value;
00309                                 break;
00310                             case 1:
00311                                 A2 = value;
00312                                 break;
00313                             case 2:
00314                                 A3 = value;
00315                                 break;
00316                             case 3:
00317                                 A4 = value;
00318                                 break;
00319                             case 4:
00320                                 A5 = value;
00321                                 break;
00322                             default:
00323                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00324                         }
00325                         break;
00326
00327                     default:
00328                         throw new ArgumentOutOfRangeException(nameof(y), y, "Coordinate out of
      range!");
```

```
00329                 }
00330             }
00331         }
00332 
00333 /// <summary>
00334 /// A <see cref="ColourMatrix"/> that whose output colour is always the same as the input colour.
00335 /// </summary>
00336         public static ColourMatrix Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0,
      1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00337 
00338 /// <summary>
00339 /// A <see cref="ColourMatrix"/> that transforms every colour in a shade of grey with approximately
      the same luminance.
00340 /// </summary>
00341         public static ColourMatrix GreyScale = new ColourMatrix(new double[,] { { 0.2126, 0.7152,
      0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 1, 0 },
      { 0, 0, 0, 0, 1 } });
00342 
00343 /// <summary>
00344 /// A <see cref="ColourMatrix"/> producing a "pastel" (desaturation) effect.
00345 /// </summary>
00346         public static ColourMatrix Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0
      }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00347 
00348 /// <summary>
00349 /// A <see cref="ColourMatrix"/> that inverts every colour, leaving the alpha component intact.
00350 /// </summary>
00351         public static ColourMatrix Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, {
      0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00352 
00353 /// <summary>
00354 /// A <see cref="ColourMatrix"/> that inverts the alpha component, leaving the other components
      intact.
00355 /// </summary>
00356         public static ColourMatrix AlphaInversion = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0
      }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 } });
00357 
00358 /// <summary>
00359 /// A <see cref="ColourMatrix"/> that shifts every colour component by an amount corresponding to the
      inverted alpha value.  The alpha component is left intact.
00360 /// </summary>
00361         public static ColourMatrix InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0,
      -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00362 
00363 /// <summary>
00364 /// Creates a <see cref="ColourMatrix"/> that turns every colour to which it is applied into the
      specified <paramref name="colour"/>.
00365 /// </summary>
00366 /// <param name="colour">The colour that will be produced by the <see cref="ColourMatrix"/>.</param>
00367 /// <param name="useAlpha">If this is <see langword="true"/>, all output pixels will have the same
      alpha value as the supplied <paramref name="colour"/>.  If this is false, the alpha value of the input
      pixels is preserved.</param>
00368 /// <returns>A <see cref="ColourMatrix"/> that turns every colour to which it is applied into the
      specified <paramref name="colour"/>.</returns>
00369         public static ColourMatrix ToColour(Colour colour, bool useAlpha = false)
00370         {
00371             if (!useAlpha)
00372             {
00373                 return new ColourMatrix(new double[,] { { 0, 0, 0, 0, colour.R }, { 0, 0, 0, 0,
      colour.G }, { 0, 0, 0, 0, colour.B }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00374             }
00375             else
00376             {
00377                 return new ColourMatrix(new double[,] { { 0, 0, 0, 0, colour.R }, { 0, 0, 0, 0,
      colour.G }, { 0, 0, 0, 0, colour.B }, { 0, 0, 0, 0, colour.A }, { 0, 0, 0, 0, 1 } });
00378             }
00379         }
00380 
00381 /// <summary>
00382 /// Creates a <see cref="ColourMatrix"/> that turns every colour to which it is applied into a shade
      of the specified <paramref name="colour"/>.  The brightness of the output colour depends on the
      luminance of the input colour.
00383 /// </summary>
00384 /// <param name="colour">The colour whose shades will be produced by the <see
      cref="ColourMatrix"/>.</param>
00385 /// <param name="useAlpha">If this is <see langword="true"/>, the transformation will also be applied
      to the alpha channel.  If this is false, the alpha value of the input pixels is preserved.</param>
00386 /// <returns>A <see cref="ColourMatrix"/> that turns every colour to which it is applied into a shade
      of the specified <paramref name="colour"/>.</returns>
00387         public static ColourMatrix LuminanceToColour(Colour colour, bool useAlpha = false)
00388         {
00389             if (!useAlpha)
00390             {
00391                 return new ColourMatrix(new double[,] { { 0.2126 * colour.R, 0.7152 * colour.R, 0.0722
      * colour.R, 0, 0 }, { 0.2126 * colour.G, 0.7152 * colour.G, 0.0722 * colour.G, 0, 0 }, { 0.2126 *
      colour.B, 0.7152 * colour.B, 0.0722 * colour.B, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00392             }
```

```
00393             else
00394             {
00395                 return new ColourMatrix(new double[,] { { 0.2126 * colour.R, 0.7152 * colour.R, 0.0722 * colour.R, 0, 0 }, { 0.2126 * colour.G, 0.7152 * colour.G, 0.0722 * colour.G, 0, 0 }, { 0.2126 * colour.B, 0.7152 * colour.B, 0.0722 * colour.B, 0, 0 }, { 0.2126 * colour.A, 0.7152 * colour.A, 0.0722 * colour.A, 0, 0 }, { 0, 0, 0, 0, 1 } });
00396             }
00397         }
00398
00399 /// <summary>
00400 /// Creates a <see cref="ColourMatrix"/> that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.
00401 /// </summary>
00402 /// <param name="preserveColour">If this is <see langword="true"/>, the values of the red, green and blue components of the input colour are preserved in the output colour.  If this is false, the output colour will always be black.</param>
00403 /// <returns>A <see cref="ColourMatrix"/> that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.</returns>
00404         public static ColourMatrix LuminanceToAlpha(bool preserveColour = false)
00405         {
00406             if (!preserveColour)
00407             {
00408                 return new ColourMatrix(new double[,] { { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 0, 1 } });
00409             }
00410             else
00411             {
00412                 return new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 0, 1 } });
00413             }
00414         }
00415
00416 /// <summary>
00417 /// Creates a new <see cref="ColourMatrix"/> whose alpha coefficients are multiplied by the specified value.
00418 /// </summary>
00419 /// <param name="alpha">The value that will be used to multiply all the alpha coefficients of the <see cref="ColourMatrix"/>.</param>
00420 /// <returns>A new <see cref="ColourMatrix"/> whose alpha coefficients have been multiplied by the specified value.</returns>
00421         public ColourMatrix WithAlpha(double alpha)
00422         {
00423             return new ColourMatrix(new double[,] { { R1, R2, R3, R4, R5 }, { G1, G2, G3, G4, G5 }, { B1, B2, B3, B4, B5 }, { A1 * alpha, A2 * alpha, A3 * alpha, A4 * alpha, A5 * alpha }, { 0, 0, 0, 0, 1 } });
00424         }
00425
00426 /// <summary>
00427 /// Concatenates two matrices.  The resulting <see cref="ColourMatrix"/> is equivalent to first applying <paramref name="matrix2"/>, and then <paramref name="matrix1"/>.
00428 /// </summary>
00429 /// <param name="matrix1">The matrix that acts second.</param>
00430 /// <param name="matrix2">The matrix that acts first.</param>
00431 /// <returns>A <see cref="ColourMatrix"/> equivalent to first applying <paramref name="matrix2"/>, and then <paramref name="matrix1"/>.</returns>
00432         public static ColourMatrix operator *(ColourMatrix matrix1, ColourMatrix matrix2)
00433         {
00434             double[,] tbr = new double[5, 5];
00435
00436             for (int i = 0; i < 5; i++)
00437             {
00438                 for (int j = 0; j < 5; j++)
00439                 {
00440                     for (int k = 0; k < 5; k++)
00441                     {
00442                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00443                     }
00444                 }
00445             }
00446
00447             return new ColourMatrix(tbr);
00448         }
00449
00450 /// <summary>
00451 /// Creates a new <see cref="ColourMatrix"/> with the specified coefficients.
00452 /// </summary>
00453 /// <param name="matrix">The coefficients of the <see cref="ColourMatrix"/>.</param>
00454         public ColourMatrix(double[,] matrix)
00455         {
00456             for (int i = 0; i < 4; i++)
00457             {
00458                 for (int j = 0; j < 5; j++)
00459                 {
00460                     this[i, j] = matrix[i, j];
00461                 }
00462             }
00463         }
```

```
00464
00465 /// <summary>
00466 /// Applies the <see cref="ColourMatrix"/> to the specified <see cref="Colour"/>.
00467 /// </summary>
00468 /// <param name="colour">The <see cref="Colour"/> to which the <see cref="ColourMatrix"/> should be
      applied.</param>
00469 /// <returns>The result of applying the <see cref="ColourMatrix"/> to the specified colour.</returns>
00470         public Colour Apply(Colour colour)
00471         {
00472             double[] col = new double[] { colour.R, colour.G, colour.B, colour.A, 1 };
00473
00474             double[] tbr = new double[5];
00475
00476             for (int i = 0; i < tbr.Length; i++)
00477             {
00478                 for (int j = 0; j < tbr.Length; j++)
00479                 {
00480                     tbr[i] += this[i, j] * col[j];
00481                 }
00482             }
00483
00484             return Colour.FromRgba(tbr[0], tbr[1], tbr[2], tbr[3]);
00485         }
00486
00487 /// <summary>
00488 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as four bytes, and
      stores the resulting colour in the
00489 /// same variables as the original RGBA values.
00490 /// </summary>
00491 /// <param name="R">The R component of the input colour.  After this method returns, this will contain
      the R component of the output colour.</param>
00492 /// <param name="G">The G component of the input colour.  After this method returns, this will contain
      the G component of the output colour.</param>
00493 /// <param name="B">The B component of the input colour.  After this method returns, this will contain
      the B component of the output colour.</param>
00494 /// <param name="A">The A component of the input colour.  After this method returns, this will contain
      the A component of the output colour.</param>
00495         public void Apply(ref byte R, ref byte G, ref byte B, ref byte A)
00496         {
00497             byte r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B *
      this.R3 + A * this.R4 + this.R5 * 255)));
00498             byte g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B *
      this.G3 + A * this.G4 + this.G5 * 255)));
00499             byte b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B *
      this.B3 + A * this.B4 + this.B5 * 255)));
00500             byte a = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.A1 + G * this.A2 + B *
      this.A3 + A * this.A4 + this.A5 * 255)));
00501
00502             R = r;
00503             G = g;
00504             B = b;
00505             A = a;
00506         }
00507
00508 /// <summary>
00509 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as three bytes, and
      stores the resulting colour in the
00510 /// same variables as the original RGB values.
00511 /// </summary>
00512 /// <param name="R">The R component of the input colour.  After this method returns, this will contain
      the R component of the output colour.</param>
00513 /// <param name="G">The G component of the input colour.  After this method returns, this will contain
      the G component of the output colour.</param>
00514 /// <param name="B">The B component of the input colour.  After this method returns, this will contain
      the B component of the output colour.</param>
00515         public void Apply(ref byte R, ref byte G, ref byte B)
00516         {
00517             byte r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B *
      this.R3 + 255 * this.R4 + this.R5 * 255)));
00518             byte g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B *
      this.G3 + 255 * this.G4 + this.G5 * 255)));
00519             byte b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B *
      this.B3 + 255 * this.B4 + this.B5 * 255)));
00520
00521             R = r;
00522             G = g;
00523             B = b;
00524         }
00525
00526 /// <summary>
00527 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as four bytes, and
      stores the resulting colour in the
00528 /// specified output bytes.
00529 /// </summary>
00530 /// <param name="R">The R component of the input colour.</param>
00531 /// <param name="G">The G component of the input colour.</param>
00532 /// <param name="B">The B component of the input colour.</param>
```

```
00533 /// <param name="A">The A component of the input colour.</param>
00534 /// <param name="r">After this method returns, this will contain the R component of the output
      colour.</param>
00535 /// <param name="g">After this method returns, this will contain the G component of the output
      colour.</param>
00536 /// <param name="b">After this method returns, this will contain the B component of the output
      colour.</param>
00537 /// <param name="a">After this method returns, this will contain the A component of the output
      colour.</param>
00538         public void Apply(byte R, byte G, byte B, byte A, out byte r, out byte g, out byte b, out byte
      a)
00539         {
00540             r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B * this.R3 + A
      * this.R4 + this.R5 * 255)));
00541             g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B * this.G3 + A
      * this.G4 + this.G5 * 255)));
00542             b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B * this.B3 + A
      * this.B4 + this.B5 * 255)));
00543             a = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.A1 + G * this.A2 + B * this.A3 + A
      * this.A4 + this.A5 * 255)));
00544         }
00545
00546 /// <summary>
00547 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as three bytes, and
      stores the resulting colour in the
00548 /// specified output bytes.
00549 /// </summary>
00550 /// <param name="R">The R component of the input colour.</param>
00551 /// <param name="G">The G component of the input colour.</param>
00552 /// <param name="B">The B component of the input colour.</param>
00553 /// <param name="r">After this method returns, this will contain the R component of the output
      colour.</param>
00554 /// <param name="g">After this method returns, this will contain the G component of the output
      colour.</param>
00555 /// <param name="b">After this method returns, this will contain the B component of the output
      colour.</param>
00556         public void Apply(byte R, byte G, byte B, out byte r, out byte g, out byte b)
00557         {
00558             r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B * this.R3 +
      255 * this.R4 + this.R5 * 255)));
00559             g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B * this.G3 +
      255 * this.G4 + this.G5 * 255)));
00560             b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B * this.B3 +
      255 * this.B4 + this.B5 * 255)));
00561         }
00562     }
00563
00564 /// <summary>
00565 /// Represents a filter that applies a <see cref="Filters.ColourMatrix"/> to the colours of the image.
00566 /// </summary>
00567     public class ColourMatrixFilter :  ILocationInvariantFilter
00568     {
00569 /// <summary>
00570 /// The <see cref="Filters.ColourMatrix"/> that is applied by this filter.
00571 /// </summary>
00572         public ColourMatrix ColourMatrix { get; }
00573
00574 /// <inheritdoc/>
00575         public Point TopLeftMargin { get; } = new Point();
00576
00577 /// <inheritdoc/>
00578         public Point BottomRightMargin { get; } = new Point();
00579
00580 /// <summary>
00581 /// Creates a new <see cref="ColourMatrixFilter"/> with the specified <see
      cref="Filters.ColourMatrix"/>.
00582 /// </summary>
00583 /// <param name="colorMatrix">The <see cref="Filters.ColourMatrix"/> that will be applied by the
      filter.</param>
00584         public ColourMatrixFilter(ColourMatrix colorMatrix)
00585         {
00586             this.ColourMatrix = colorMatrix;
00587         }
00588
00589 /// <inheritdoc/>
00590         public RasterImage Filter(RasterImage image, double scale)
00591         {
00592             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00593             GC.AddMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00594
00595             int width = image.Width;
00596             int height = image.Height;
00597
00598             int pixelSize = image.HasAlpha ?  4 :  3;
00599             int stride = image.Width * pixelSize;
00600
```

```
00601             int threads = Math.Min(8, Environment.ProcessorCount);
00602 
00603             unsafe
00604             {
00605                 byte* input = (byte*)image.ImageDataAddress;
00606                 byte* output = (byte*)tbrData;
00607 
00608                 Action<int> yLoop;
00609 
00610                 if (image.HasAlpha)
00611                 {
00612                     yLoop = (y) =>
00613                     {
00614                         for (int x = 0; x < width; x++)
00615                         {
00616                             this.ColourMatrix.Apply(input[y * stride + x * 4], input[y * stride + x *
      4 + 1], input[y * stride + x * 4 + 2], input[y * stride + x * 4 + 3], out output[y * stride + x * 4],
      out output[y * stride + x * 4 + 1], out output[y * stride + x * 4 + 2], out output[y * stride + x * 4
      + 3]);
00617                         }
00618                     };
00619                 }
00620                 else
00621                 {
00622                     yLoop = (y) =>
00623                     {
00624                         for (int x = 0; x < width; x++)
00625                         {
00626                             this.ColourMatrix.Apply(input[y * stride + x * 4], input[y * stride + x *
      4 + 1], input[y * stride + x * 4 + 2], out output[y * stride + x * 4], out output[y * stride + x * 4 +
      1], out output[y * stride + x * 4 + 2]);
00627                         }
00628                     };
00629                 }
00630 
00631                 if (threads == 1)
00632                 {
00633                     for (int y = 0; y < height; y++)
00634                     {
00635                         yLoop(y);
00636                     }
00637                 }
00638                 else
00639                 {
00640                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00641 
00642                     Parallel.For(0, height, options, yLoop);
00643                 }
00644 
00645             }
00646 
00647             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00648 
00649             return new RasterImage(ref disp, width, height, image.HasAlpha, image.Interpolate);
00650         }
00651     }
00652 }
```

8.57 CompositeFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System.Collections.Generic;
00019 using System.Collections.Immutable;
00020 
00021 namespace VectSharp.Filters
00022 {
```

```
00023 /// <summary>
00024 /// Represents a filter that corresponds to applying multiple <see cref="ILocationInvariantFilter"/>s
      one after the other.
00025 /// </summary>
00026     public class CompositeLocationInvariantFilter :  ILocationInvariantFilter
00027     {
00028 /// <inheritdoc/>
00029         public Point TopLeftMargin { get; }
00030
00031 /// <inheritdoc/>
00032         public Point BottomRightMargin { get; }
00033
00034 /// <summary>
00035 /// The filters that are applied by this filter.
00036 /// </summary>
00037         public ImmutableList<ILocationInvariantFilter> Filters { get; }
00038
00039 /// <summary>
00040 /// Creates a new <see cref="CompositeLocationInvariantFilter"/> with the specified filters.
00041 /// </summary>
00042 /// <param name="filters">The filters that will be applied by the new filter.</param>
00043         public CompositeLocationInvariantFilter(IEnumerable<ILocationInvariantFilter> filters)
00044         {
00045             IEnumerable<ILocationInvariantFilter> flattenedFilters = FlattenFilters(filters);
00046             this.Filters = ImmutableList.CreateRange(flattenedFilters);
00047
00048             bool initialised = false;
00049
00050             foreach (IFilter filter in flattenedFilters)
00051             {
00052                 if (!initialised)
00053                 {
00054                     this.TopLeftMargin = filter.TopLeftMargin;
00055                     this.BottomRightMargin = filter.BottomRightMargin;
00056                     initialised = true;
00057                 }
00058                 else
00059                 {
00060                     this.TopLeftMargin = Point.Max(this.TopLeftMargin, filter.TopLeftMargin);
00061                     this.BottomRightMargin = Point.Max(this.BottomRightMargin,
      filter.BottomRightMargin);
00062                 }
00063             }
00064         }
00065
00066 /// <summary>
00067 /// Creates a new <see cref="CompositeLocationInvariantFilter"/> with the specified filters.
00068 /// </summary>
00069 /// <param name="filters">The filters that will be applied by the new filter.</param>
00070         public CompositeLocationInvariantFilter(params ILocationInvariantFilter[] filters) :
      this((IEnumerable<ILocationInvariantFilter>)filters) {  }
00071
00072         private IEnumerable<ILocationInvariantFilter>
      FlattenFilters(IEnumerable<ILocationInvariantFilter> filters)
00073         {
00074             foreach (ILocationInvariantFilter filter in filters)
00075             {
00076                 if (filter is CompositeLocationInvariantFilter composite)
00077                 {
00078                     foreach (ILocationInvariantFilter filter2 in FlattenFilters(composite.Filters))
00079                     {
00080                         yield return filter2;
00081                     }
00082                 }
00083                 else
00084                 {
00085                     yield return filter;
00086                 }
00087             }
00088         }
00089
00090 /// <inheritdoc/>
00091         public RasterImage Filter(RasterImage image, double scale)
00092         {
00093             RasterImage currImage = image;
00094
00095             foreach (ILocationInvariantFilter filter in this.Filters)
00096             {
00097                 RasterImage prevImage = currImage;
00098                 currImage = filter.Filter(prevImage, scale);
00099
00100                 if (prevImage != image)
00101                 {
00102                     prevImage.Dispose();
00103                 }
00104             }
00105
```

```
00106             return currImage;
00107         }
00108     }
00109 }
```

8.58 ConvolutionFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Threading.Tasks;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a filter that applies a matrix convolution to the image.
00025 /// </summary>
00026     public class ConvolutionFilter :  ILocationInvariantFilter
00027     {
00028 /// <inheritdoc/>
00029         public Point TopLeftMargin { get; protected set; }
00030 /// <inheritdoc/>
00031         public Point BottomRightMargin { get; protected set; }
00032
00033 /// <summary>
00034 /// The kernel of the <see cref="ConvolutionFilter"/>.  The dimensions of this matrix should all be
      odd numbers.  The larger the kernel, the worse the performance.
00035 /// </summary>
00036         public virtual double[,] Kernel { get; protected set; }
00037
00038 /// <summary>
00039 /// The normalisation value that is applies to the kernel.
00040 /// </summary>
00041         public virtual double Normalisation { get; protected set; } = 1;
00042
00043 /// <summary>
00044 /// The bias value that is added to every colour component when the filter is applied.
00045 /// </summary>
00046         public virtual double Bias { get; protected set; } = 0;
00047
00048 /// <summary>
00049 /// The scale relating the size of the kernel to graphics units.
00050 /// </summary>
00051         public virtual double Scale { get; protected set; }
00052
00053 /// <summary>
00054 /// If this is <see langword="true"/>, the alpha value of the input pixels is preserved.  Otherwise,
      the alpha channel is subject to the same convolution process as the other colour components.
00055 /// </summary>
00056         public virtual bool PreserveAlpha { get; protected set; } = true;
00057
00058 /// <summary>
00059 /// Creates a new <see cref="ConvolutionFilter"/> with the specified parameters.
00060 /// </summary>
00061 /// <param name="kernel">The kernel of the <see cref="ConvolutionFilter"/>.  The dimensions of this
      matrix should all be odd numbers.  The larger the kernel, the worse the performance.</param>
00062 /// <param name="scale">The scale relating the size of the kernel to graphics units.</param>
00063 /// <param name="preserveAlpha">If this is <see langword="true"/>, the alpha value of the input pixels
      is preserved.  Otherwise, the alpha channel is subject to the same convolution process as the other
      colour components.</param>
00064 /// <param name="normalisation">The normalisation value that is applies to the kernel.</param>
00065 /// <param name="bias">The bias value that is added to every colour component when the filter is
      applied.</param>
00066 /// <exception cref="ArgumentException">This exception is thrown when the kernel dimensions are not
      odd numbers.</exception>
00067         public ConvolutionFilter(double[,] kernel, double scale, bool preserveAlpha = true, double
      normalisation = 1, double bias = 0)
00068         {
```

```
00069             if (kernel.GetLength(0) % 2 != 1 || kernel.GetLength(1) % 2 != 1)
00070             {
00071                 throw new ArgumentException("The kernel must have an odd number of rows and columns!",
      nameof(kernel));
00072             }
00073
00074             this.Kernel = kernel;
00075
00076             int kernelWidth = (this.Kernel.GetLength(0) - 1) / 2;
00077             int kernelHeight = (this.Kernel.GetLength(1) - 1) / 2;
00078
00079             this.TopLeftMargin = new Point(kernelWidth * scale, kernelHeight * scale);
00080             this.BottomRightMargin = new Point(kernelWidth * scale, kernelHeight * scale);
00081             this.Scale = scale;
00082             this.PreserveAlpha = preserveAlpha;
00083             this.Normalisation = normalisation;
00084             this.Bias = bias;
00085         }
00086
00087 /// <inheritdoc/>
00088         public virtual RasterImage Filter(RasterImage image, double scale)
00089         {
00090             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00091
00092             int width = image.Width;
00093             int height = image.Height;
00094
00095             int kernelWidth = (this.Kernel.GetLength(0) - 1) / 2;
00096             int kernelHeight = (this.Kernel.GetLength(1) - 1) / 2;
00097
00098             int actualKernelWidth = (int)Math.Round(kernelWidth * scale * this.Scale);
00099             int actualKernelHeight = (int)Math.Round(kernelHeight * scale * this.Scale);
00100
00101             actualKernelWidth = Math.Max(actualKernelWidth, 1);
00102             actualKernelHeight = Math.Max(actualKernelHeight, 1);
00103
00104             int[] kernelX = new int[actualKernelWidth * 2 + 1];
00105
00106             for (int x = 0; x < actualKernelWidth * 2 + 1; x++)
00107             {
00108                 kernelX[x] = (int)Math.Round((double)(x - actualKernelWidth) / actualKernelWidth *
      kernelWidth + kernelWidth);
00109             }
00110
00111             int[] kernelY = new int[actualKernelHeight * 2 + 1];
00112
00113             for (int y = 0; y < actualKernelHeight * 2 + 1; y++)
00114             {
00115                 kernelY[y] = (int)Math.Round((double)(y - actualKernelHeight) / actualKernelHeight *
      kernelHeight + kernelHeight);
00116             }
00117
00118             int[] countsX = new int[2 * kernelWidth + 1];
00119
00120             for (int i = 0; i < 2 * actualKernelWidth + 1; i++)
00121             {
00122                 countsX[kernelX[i]]++;
00123             }
00124
00125             int[] countsY = new int[2 * kernelHeight + 1];
00126
00127             for (int i = 0; i < 2 * actualKernelHeight + 1; i++)
00128             {
00129                 countsY[kernelY[i]]++;
00130             }
00131
00132
00133             double[] weightsX = new double[2 * actualKernelWidth + 1];
00134
00135             for (int i = 0; i < 2 * actualKernelWidth + 1; i++)
00136             {
00137                 weightsX[i] = 1.0 / countsX[kernelX[i]];
00138             }
00139
00140             double[] weightsY = new double[2 * actualKernelHeight + 1];
00141
00142             for (int i = 0; i < 2 * actualKernelHeight + 1; i++)
00143             {
00144                 weightsY[i] = 1.0 / countsY[kernelY[i]];
00145             }
00146
00147             kernelWidth = actualKernelWidth;
00148             kernelHeight = actualKernelHeight;
00149
00150             double normalisation = this.Normalisation;
00151
```

```
00152             if (double.IsNaN(normalisation) || normalisation == 0)
00153             {
00154                 normalisation = 1;
00155             }
00156 
00157             double totalWeight = 0;
00158             for (int i = 0; i < kernelWidth * 2 + 1; i++)
00159             {
00160                 for (int j = 0; j < kernelHeight * 2 + 1; j++)
00161                 {
00162                     totalWeight += Kernel[kernelX[i], kernelY[j]] * weightsX[i] * weightsY[j];
00163                 }
00164             }
00165 
00166             if (Math.Abs(totalWeight) < 1e-5)
00167             {
00168                 totalWeight = 1;
00169             }
00170 
00171 
00172             double bias = this.Bias * 255;
00173             int pixelSize = image.HasAlpha ?  4 :  3;
00174             int stride = image.Width * pixelSize;
00175 
00176             int threads = Math.Min(8, Environment.ProcessorCount);
00177 
00178             unsafe
00179             {
00180                 byte* input = (byte*)image.ImageDataAddress;
00181                 byte* output = (byte*)tbrData;
00182 
00183                 Action<int> yLoop;
00184 
00185                 if (image.HasAlpha && !PreserveAlpha)
00186                 {
00187                     yLoop = (y) =>
00188                     {
00189                         for (int x = 0; x < width; x++)
00190                         {
00191                             double R = 0;
00192                             double G = 0;
00193                             double B = 0;
00194 
00195                             double weight = 0;
00196 
00197                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00198                             {
00199                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00200                                 {
00201                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00202                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00203 
00204                                     double a = input[tY * stride + tX * 4 + 3] / 255.0 *
      weightsX[targetX] * weightsY[targetY];
00205 
00206                                     int projectedX = kernelX[targetX];
00207                                     int projectedY = kernelY[targetY];
00208 
00209                                     weight += Kernel[projectedX, projectedY] * a;
00210 
00211                                     R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4]
      * a;
00212                                     G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      1] * a;
00213                                     B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      2] * a;
00214                                 }
00215                             }
00216 
00217                             if (weight != 0)
00218                             {
00219                                 output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * weight) + bias));
00220                                 output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * weight) + bias));
00221                                 output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * weight) + bias));
00222                                 output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      (weight / (normalisation * totalWeight) * 255 + bias)));
00223                             }
00224                             else
00225                             {
00226                                 output[y * stride + x * 4] = 0;
00227                                 output[y * stride + x * 4 + 1] = 0;
00228                                 output[y * stride + x * 4 + 2] = 0;
```

```
00229                                 output[y * stride + x * 4 + 3] = 0;
00230                             }
00231                         }
00232                     };
00233                 }
00234                 else if (image.HasAlpha && PreserveAlpha)
00235                 {
00236                     yLoop = (y) =>
00237                     {
00238                         for (int x = 0; x < width; x++)
00239                         {
00240                             double R = 0;
00241                             double G = 0;
00242                             double B = 0;
00243 
00244                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00245                             {
00246                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00247                                 {
00248                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00249                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00250 
00251                                     int projectedX = kernelX[targetX];
00252                                     int projectedY = kernelY[targetY];
00253 
00254                                     R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4]
      * weightsX[targetX] * weightsY[targetY];
00255                                     G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      1] * weightsX[targetX] * weightsY[targetY];
00256                                     B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      2] * weightsX[targetX] * weightsY[targetY];
00257                                 }
00258                             }
00259 
00260                             output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * totalWeight) + bias));
00261                             output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * totalWeight) + bias));
00262                             output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * totalWeight) + bias));
00263                             output[y * stride + x * 4 + 3] = input[y * stride + x * 4 + 3];
00264                         }
00265                     };
00266                 }
00267                 else
00268                 {
00269                     yLoop = (y) =>
00270                     {
00271                         for (int x = 0; x < width; x++)
00272                         {
00273                             double R = 0;
00274                             double G = 0;
00275                             double B = 0;
00276 
00277                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00278                             {
00279                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00280                                 {
00281                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00282                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00283 
00284                                     int projectedX = kernelX[targetX];
00285                                     int projectedY = kernelY[targetY];
00286 
00287                                     R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3]
      * weightsX[targetX] * weightsY[targetY];
00288                                     G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3 +
      1] * weightsX[targetX] * weightsY[targetY];
00289                                     B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3 +
      2] * weightsX[targetX] * weightsY[targetY];
00290                                 }
00291                             }
00292 
00293                             output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * totalWeight) + bias));
00294                             output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * totalWeight) + bias));
00295                             output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * totalWeight) + bias));
00296                         }
00297                     };
00298                 }
00299 
```

```
00300                 if (threads == 1)
00301                 {
00302                     for (int y = 0; y < height; y++)
00303                     {
00304                         yLoop(y);
00305                     }
00306                 }
00307                 else
00308                 {
00309                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00310 
00311                     Parallel.For(0, height, options, yLoop);
00312                 }
00313             }
00314 
00315             return new RasterImage(tbrData, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
00316         }
00317     }
00318 }
```

8.59 Filters.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 
00020 namespace VectSharp.Filters
00021 {
00022 /// <summary>
00023 /// Represents a filter.  Do not implement this interface directly; instead, implement <see
      cref="ILocationInvariantFilter"/> or <see cref="IFilterWithLocation"/>.
00024 /// </summary>
00025     public interface IFilter
00026     {
00027 /// <summary>
00028 /// Determines how much the area of the filter's subject should be expanded on the top-left to
      accommodate the results of the filter.
00029 /// </summary>
00030         Point TopLeftMargin { get; }
00031 
00032 /// <summary>
00033 /// Determines how much the area of the filter's subject should be expanded on the bottom-right to
      accommodate the results of the filter.
00034 /// </summary>
00035         Point BottomRightMargin { get; }
00036     }
00037 
00038 /// <summary>
00039 /// Represents a filter that can be applied to an image regardless of its location on the graphics
      surface.
00040 /// </summary>
00041     public interface ILocationInvariantFilter :  IFilter
00042     {
00043 /// <summary>
00044 /// Applies the filter to a <see cref="RasterImage"/>.
00045 /// </summary>
00046 /// <param name="image">The <see cref="RasterImage"/> to which the filter will be applied.</param>
00047 /// <param name="scale">The scale of the image with respect to the filter.</param>
00048 /// <returns>A new <see cref="RasterImage"/> containing the filtered image.  The source <paramref
      name="image"/> is left unaltered.</returns>
00049         RasterImage Filter(RasterImage image, double scale);
00050     }
00051 
00052 /// <summary>
00053 /// Represents a filter whose results depend on the position of the subject image on the graphics
      surface.
```

```
00054 /// </summary>
00055     public interface IFilterWithLocation :  IFilter
00056     {
00057 /// <summary>
00058 /// Applies the filter to a <see cref="RasterImage"/>.
00059 /// </summary>
00060 /// <param name="image">The <see cref="RasterImage"/> to which the filter will be applied.</param>
00061 /// <param name="bounds">The region on the graphics surface where the image will be drawn.</param>
00062 /// <param name="scale">The scale of the image with respect to the filter.</param>
00063 /// <returns>A new <see cref="RasterImage"/> containing the filtered image.  The source <paramref
      name="image"/> is left unaltered.</returns>
00064         RasterImage Filter(RasterImage image, Rectangle bounds, double scale);
00065     }
00066
00067 /// <summary>
00068 /// Represents a filter with a parameter that needs to be rasterised at the same resolution as the
      subject image prior to applying the filter.
00069 /// The <see cref="FilterWithRasterisableParameter"/> abstract class provides a default implementation
      of this interface.
00070 /// </summary>
00071     public interface IFilterWithRasterisableParameter
00072     {
00073 /// <summary>
00074 /// Rasterises the filter's parameter at the specified scale, using the specified rasterisation
      method.
00075 /// </summary>
00076 /// <param name="rasterisationMethod">The method used to rasterise the image.  The first argument of
      this method is the <see cref="Graphics"/> to be rasterised,
00077 /// the second is a <see cref="Rectangle"/> representing the region to rasterise, the third is a <see
      cref="double"/> representing the scale, and the third is
00078 /// a boolean value indicating whether the resulting <see cref="RasterImage"/> should be
      interpolated.</param>
00079 /// <param name="scale">The scale factor at which the parameter is rasterised.</param>
00080         void RasteriseParameter(Func<Graphics, Rectangle, double, bool, RasterImage>
      rasterisationMethod, double scale);
00081     }
00082
00083 /// <summary>
00084 /// Represents a filter with a parameter that needs to be rasterised at the same resolution as the
      subject image prior to applying the filter.
00085 /// </summary>
00086     public abstract class FilterWithRasterisableParameter :  IFilterWithRasterisableParameter,
      IDisposable
00087     {
00088 /// <summary>
00089 /// The parameter that needs to be rasterised.
00090 /// </summary>
00091         protected virtual Graphics RasterisableParameter { get; }
00092
00093 /// <summary>
00094 /// The result of the last rasterisation of the <see cref="RasterisableParameter"/>.
00095 /// </summary>
00096         protected RasterImage cachedRasterisation;
00097         private bool disposedValue;
00098
00099 /// <summary>
00100 /// Get a rasterised version of the <see cref="RasterisableParameter"/> at the specified scale.  If
      this has already been computed, the cached result is returned.
00101 /// Otherwise, the image is rasterised using the default rasterisation method.
00102 /// </summary>
00103 /// <param name="scale">The scale at which the <see cref="RasterisableParameter"/> will be
      rasterised.</param>
00104 /// <returns>A rasterised version of the <see cref="RasterisableParameter"/> at the specified
      scale.</returns>
00105         protected virtual RasterImage GetCachedRasterisation(double scale)
00106         {
00107             if (this.cachedRasterisation == null || CachedResolution != scale)
00108             {
00109                 this.RasteriseParameter(scale);
00110             }
00111
00112             return this.cachedRasterisation;
00113         }
00114
00115 /// <summary>
00116 /// The bounds of the last cached rendering of the <see cref="RasterisableParameter"/> on the graphics
      surface.
00117 /// </summary>
00118         protected virtual Rectangle CachedBounds { get; set; }
00119
00120 /// <summary>
00121 /// The resolution using which the cached rendering of the <see cref="RasterisableParameter"/> has
      been computed.
00122 /// </summary>
00123         protected virtual double CachedResolution { get; set; } = double.NaN;
00124
00125 /// <summary>
```

```
00126 /// Create a new <see cref="FilterWithRasterisableParameter"/> with the specified parameter.
00127 /// </summary>
00128 /// <param name="rasterisableParameter">The parameter that needs to be rasterised at the same
      resolution as the subject image prior to applying the filter.</param>
00129         protected FilterWithRasterisableParameter(Graphics rasterisableParameter)
00130         {
00131             this.RasterisableParameter = rasterisableParameter;
00132         }
00133 
00134 /// <inheritdoc />
00135         public virtual void RasteriseParameter(Func<Graphics, Rectangle, double, bool, RasterImage>
      rasterisationMethod, double scale)
00136         {
00137             Rectangle bounds = RasterisableParameter.GetBounds();
00138 
00139             this.cachedRasterisation?.Dispose();
00140 
00141             this.cachedRasterisation = rasterisationMethod(this.RasterisableParameter, bounds, scale,
      true);
00142             this.CachedResolution = scale;
00143             this.CachedBounds = bounds;
00144         }
00145 
00146 /// <summary>
00147 /// Rasterises the filter's parameter at the specified scale, using the default rasterisation method.
00148 /// </summary>
00149 /// <param name="scale">The scale factor at which the parameter is rasterised.</param>
00150 /// <exception cref="NotImplementedException">This exception is thrown when there is no default
      rasterisation method.  This occurs because neither the VectSharp.Raster
00151 /// assembly, nor the VectSharp.Raster.ImageSharp assembly have been loaded, and no custom
      implementation of <see cref="Graphics.RasterisationMethod"/> has been provided.</exception>
00152         protected virtual void RasteriseParameter(double scale)
00153         {
00154             Rectangle bounds = this.RasterisableParameter.GetBounds();
00155 
00156             this.cachedRasterisation?.Dispose();
00157 
00158             if (this.RasterisableParameter.TryRasterise(bounds, scale, true, out RasterImage raster))
00159             {
00160                 this.cachedRasterisation = raster;
00161                 this.CachedResolution = scale;
00162                 this.CachedBounds = bounds;
00163             }
00164             else
00165             {
00166                 throw new NotImplementedException(@"The filter could not be rasterised!  You can avoid
      this error by doing one of the following:
00167 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
00168 • Provide your own implementation of Graphics.RasterisationMethod.");
00169             }
00170         }
00171 
00172 /// <inheritdoc/>
00173         protected virtual void Dispose(bool disposing)
00174         {
00175             if (!disposedValue)
00176             {
00177                 if (disposing)
00178                 {
00179                     this.cachedRasterisation?.Dispose();
00180                 }
00181 
00182                 disposedValue = true;
00183             }
00184         }
00185 
00186 /// <inheritdoc/>
00187         public void Dispose()
00188         {
00189             Dispose(disposing:  true);
00190             GC.SuppressFinalize(this);
00191         }
00192     }
00193 
00194 
00195 }
```

8.60 GaussianBlurFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
```

```
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Threading.Tasks;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a filter that applies a Gaussian blur effect.
00025 /// </summary>
00026     public class GaussianBlurFilter :  ILocationInvariantFilter
00027     {
00028 /// <summary>
00029 /// The standard deviation of the Gaussian blur.
00030 /// </summary>
00031         public double StandardDeviation { get; }
00032
00033 /// <inheritdoc/>
00034         public Point TopLeftMargin { get; }
00035
00036 /// <inheritdoc/>
00037         public Point BottomRightMargin { get; }
00038
00039 /// <summary>
00040 /// Creates a new <see cref="GaussianBlurFilter"/> with the specified standard deviation.
00041 /// </summary>
00042 /// <param name="standardDeviation">The standard deviation of the Gaussian blur.</param>
00043         public GaussianBlurFilter(double standardDeviation)
00044         {
00045             this.StandardDeviation = standardDeviation;
00046             this.TopLeftMargin = new Point(standardDeviation * 3, standardDeviation * 3);
00047             this.BottomRightMargin = new Point(standardDeviation * 3, standardDeviation * 3);
00048         }
00049
00050 /// <inheritdoc/>
00051         public RasterImage Filter(RasterImage image, double scale)
00052         {
00053             return FilterSRGB(image, scale);
00054         }
00055
00056         private RasterImage FilterSRGB(RasterImage image, double scale)
00057         {
00058             if (this.StandardDeviation * scale > 2)
00059             {
00060                 int[] boxes = BoxesForGauss(this.StandardDeviation * scale, 3);
00061
00062                 for (int i = 0; i < boxes.Length; i++)
00063                 {
00064                     BoxBlurFilter box = new BoxBlurFilter((boxes[i] - 1) / 2);
00065
00066                     image = box.Filter(image, 1);
00067                 }
00068             }
00069             else
00070             {
00071                 image = FilterSRGBExact(image, this.StandardDeviation * scale);
00072             }
00073
00074             return image;
00075         }
00076
00077         // Adapted from http://blog.ivank.net/fastest-gaussian-blur.html
00078         private static int[] BoxesForGauss(double standardDeviation, int count)
00079         {
00080             double idealWeight = Math.Sqrt((12 * standardDeviation * standardDeviation / count) + 1);
00081
00082             int lowerWeight = (int)Math.Floor(idealWeight);
00083
00084             if (lowerWeight % 2 == 0)
00085             {
00086                 lowerWeight--;
00087             }
00088
00089             int upperWeight = lowerWeight + 2;
00090
```

```
00091             double idealSize = (12 * standardDeviation * standardDeviation - count * lowerWeight *
      lowerWeight - 4 * count * lowerWeight - 3 * count) / (-4 * lowerWeight - 4);
00092
00093             int m = (int)Math.Round(idealSize);
00094
00095             int[] sizes = new int[count];
00096
00097             for (int i = 0; i < count; i++)
00098             {
00099                 sizes[i] = i < m ?  lowerWeight :  upperWeight;
00100             }
00101
00102             return sizes;
00103         }
00104
00105         private RasterImage FilterSRGBExact(RasterImage image, double standardDeviation)
00106         {
00107             IntPtr intermediateData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width
      * image.Height * (image.HasAlpha ?  4 :  3));
00108             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00109             GC.AddMemoryPressure(2 * image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00110
00111             int width = image.Width;
00112             int height = image.Height;
00113
00114             int kernelSize = (int)Math.Ceiling(standardDeviation * 3);
00115
00116             double[] kernel = new double[kernelSize * 2 + 1];
00117
00118             double total = 0;
00119
00120             for (int i = 0; i <= kernelSize; i++)
00121             {
00122                 kernel[i] = Math.Exp(-(kernelSize - i) * (kernelSize - i) / (2 * standardDeviation *
      standardDeviation));
00123
00124                 if (i < kernelSize)
00125                 {
00126                     kernel[2 * kernelSize - i] = kernel[i];
00127
00128                     total += kernel[i] * 2;
00129                 }
00130                 else
00131                 {
00132                     total += kernel[i];
00133                 }
00134             }
00135
00136             for (int i = 0; i < kernel.Length; i++)
00137             {
00138                 kernel[i] /= total;
00139             }
00140
00141             int pixelSize = image.HasAlpha ?  4 :  3;
00142             int stride = image.Width * pixelSize;
00143
00144             int threads;
00145
00146             double size = Math.Sqrt((double)image.Width * image.Height);
00147
00148             if (size <= 128)
00149             {
00150                 threads = 1;
00151             }
00152             else if (size <= 512)
00153             {
00154                 threads = Math.Min(4, Environment.ProcessorCount);
00155             }
00156             else
00157             {
00158                 threads = Math.Min(8, Environment.ProcessorCount);
00159             }
00160
00161             unsafe
00162             {
00163                 byte* input = (byte*)image.ImageDataAddress;
00164                 byte* intermediate = (byte*)intermediateData;
00165                 byte* output = (byte*)tbrData;
00166
00167                 Action<int> yLoop;
00168
00169                 if (image.HasAlpha)
00170                 {
00171                     yLoop = y =>
00172                     {
00173                         for (int x = 0; x < width; x++)
```

```
00174                         {
00175                             double R = 0;
00176                             double G = 0;
00177                             double B = 0;
00178                             double weight = 0;
00179 
00180                             for (int targetX = 0; targetX <= kernelSize * 2; targetX++)
00181                             {
00182                                 int tX = Math.Min(Math.Max(0, x + targetX - kernelSize), width - 1);
00183 
00184                                 double a = input[y * stride + tX * 4 + 3] / 255.0;
00185 
00186                                 weight += kernel[targetX] * a;
00187 
00188                                 R += kernel[targetX] * input[y * stride + tX * 4] * a;
00189                                 G += kernel[targetX] * input[y * stride + tX * 4 + 1] * a;
00190                                 B += kernel[targetX] * input[y * stride + tX * 4 + 2] * a;
00191                             }
00192 
00193                             if (weight != 0)
00194                             {
00195                                 intermediate[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      weight));
00196                                 intermediate[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0,
      G / weight));
00197                                 intermediate[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0,
      B / weight));
00198                                 intermediate[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      weight * 255));
00199                             }
00200                             else
00201                             {
00202                                 intermediate[y * stride + x * 4] = 0;
00203                                 intermediate[y * stride + x * 4 + 1] = 0;
00204                                 intermediate[y * stride + x * 4 + 2] = 0;
00205                                 intermediate[y * stride + x * 4 + 3] = 0;
00206                             }
00207                         }
00208                     };
00209                 }
00210                 else
00211                 {
00212                     yLoop = y =>
00213                     {
00214                         for (int x = 0; x < width; x++)
00215                         {
00216                             double R = 0;
00217                             double G = 0;
00218                             double B = 0;
00219 
00220                             for (int targetX = 0; targetX <= kernelSize * 2; targetX++)
00221                             {
00222                                 int tX = Math.Min(Math.Max(0, x + targetX - kernelSize), width - 1);
00223 
00224                                 R += kernel[targetX] * input[y * stride + tX * 3];
00225                                 G += kernel[targetX] * input[y * stride + tX * 3 + 1];
00226                                 B += kernel[targetX] * input[y * stride + tX * 3 + 2];
00227                             }
00228 
00229                             intermediate[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R));
00230                             intermediate[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0,
      G));
00231                             intermediate[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0,
      B));
00232                         }
00233                     };
00234                 }
00235 
00236                 Action<int> xLoop;
00237 
00238                 if (image.HasAlpha)
00239                 {
00240                     xLoop = y =>
00241                     {
00242                         for (int x = 0; x < width; x++)
00243                         {
00244                             double R = 0;
00245                             double G = 0;
00246                             double B = 0;
00247 
00248 
00249                             double weight = 0;
00250 
00251                             for (int targetY = 0; targetY <= kernelSize * 2; targetY++)
00252                             {
00253                                 int tY = Math.Min(Math.Max(0, y + targetY - kernelSize), height - 1);
00254 
```

```
00255                                   double a = intermediate[tY * stride + x * 4 + 3] / 255.0;
00256
00257                                   weight += kernel[targetY] * a;
00258
00259                                   R += kernel[targetY] * intermediate[tY * stride + x * 4] * a;
00260                                   G += kernel[targetY] * intermediate[tY * stride + x * 4 + 1] * a;
00261                                   B += kernel[targetY] * intermediate[tY * stride + x * 4 + 2] * a;
00262                               }
00263
00264                               if (weight != 0)
00265                               {
00266                                   output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      weight));
00267                                   output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      weight));
00268                                   output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      weight));
00269                                   output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      weight * 255));
00270                               }
00271                               else
00272                               {
00273                                   output[y * stride + x * 4] = 0;
00274                                   output[y * stride + x * 4 + 1] = 0;
00275                                   output[y * stride + x * 4 + 2] = 0;
00276                                   output[y * stride + x * 4 + 3] = 0;
00277                               }
00278                           }
00279                       };
00280                   }
00281                   else
00282                   {
00283                       xLoop = y =>
00284                       {
00285                           for (int x = 0; x < width; x++)
00286                           {
00287                               double R = 0;
00288                               double G = 0;
00289                               double B = 0;
00290
00291                               for (int targetY = 0; targetY <= kernelSize * 2; targetY++)
00292                               {
00293                                   int tY = Math.Min(Math.Max(0, y + targetY - kernelSize), height - 1);
00294
00295                                   R += kernel[targetY] * intermediate[tY * stride + x * 3];
00296                                   G += kernel[targetY] * intermediate[tY * stride + x * 3 + 1];
00297                                   B += kernel[targetY] * intermediate[tY * stride + x * 3 + 2];
00298                               }
00299
00300                               output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R));
00301                               output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, G));
00302                               output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, B));
00303                           }
00304                       };
00305                   }
00306
00307
00308
00309                   if (threads == 1)
00310                   {
00311                       for (int y = 0; y < height; y++)
00312                       {
00313                           yLoop(y);
00314                       }
00315
00316                       for (int y = 0; y < height; y++)
00317                       {
00318                           xLoop(y);
00319                       }
00320                   }
00321                   else
00322                   {
00323                       ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00324
00325                       Parallel.For(0, height, options, yLoop);
00326                       Parallel.For(0, height, options, xLoop);
00327                   }
00328               }
00329
00330               System.Runtime.InteropServices.Marshal.FreeHGlobal(intermediateData);
00331               GC.RemoveMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00332
00333               DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00334
00335               return new RasterImage(ref disp, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
```

```
00336         }
00337
00338     }
00339 }
00340
```

8.61 MaskFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Threading.Tasks;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a filter that uses the luminance of an image to mask another image.
00025 /// </summary>
00026     public class MaskFilter :  FilterWithRasterisableParameter, IFilterWithLocation
00027     {
00028 /// <inheritdoc/>
00029         public Point TopLeftMargin => new Point(0, 0);
00030 /// <inheritdoc/>
00031         public Point BottomRightMargin => new Point(0, 0);
00032
00033 /// <summary>
00034 /// The image that is used to mask the input image.
00035 /// </summary>
00036         public Graphics Mask => this.RasterisableParameter;
00037
00038 /// <summary>
00039 /// Creates a new <see cref="MaskFilter"/> with the specified mask image.
00040 /// </summary>
00041 /// <param name="mask">The image that is used to mask the input image.</param>
00042         public MaskFilter(Graphics mask) :  base(mask) { }
00043
00044 /// <inheritdoc/>
00045         public RasterImage Filter(RasterImage image, Rectangle bounds, double scale)
00046         {
00047             RasterImage mask = this.GetCachedRasterisation(scale);
00048
00049             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
    image.Height * 4);
00050             GC.AddMemoryPressure(image.Width * image.Height * 4);
00051
00052             int width = image.Width;
00053             int height = image.Height;
00054
00055             int pixelSizeInput = image.HasAlpha ?  4 :  3;
00056             int strideInput = image.Width * pixelSizeInput;
00057
00058             int pixelSizeOutput = 4;
00059             int strideOutput = image.Width * pixelSizeOutput;
00060
00061             int pixelSizeMask = 4;
00062             int strideMask = mask.Width * pixelSizeMask;
00063
00064             int maskWidth = mask.Width;
00065             int maskHeight = mask.Height;
00066
00067             int threads = Math.Min(8, Environment.ProcessorCount);
00068
00069             unsafe
00070             {
00071                 byte* input = (byte*)image.ImageDataAddress;
00072                 byte* maskBytes = (byte*)mask.ImageDataAddress;
00073                 byte* output = (byte*)tbrData;
00074
```

```
00075                 Action<int> yLoop;
00076
00077                 if (image.HasAlpha)
00078                 {
00079                     yLoop = (y) =>
00080                     {
00081                         for (int x = 0; x < width; x++)
00082                         {
00083                             int maskX = (int)Math.Round((bounds.Location.X + (x + 0.5) *
      bounds.Size.Width / width - CachedBounds.Location.X) / CachedBounds.Size.Width * maskWidth);
00084                             int maskY = (int)Math.Round((bounds.Location.Y + (y + 0.5) *
      bounds.Size.Height / height - CachedBounds.Location.Y) / CachedBounds.Size.Height * maskHeight);
00085
00086                             double weight = 0;
00087
00088                             if (maskX >= 0 && maskX < maskWidth && maskY >= 0 && maskY < maskHeight)
00089                             {
00090                                 weight = (maskBytes[maskY * strideMask + maskX * pixelSizeMask] *
      0.2126 + maskBytes[maskY * strideMask + maskX * pixelSizeMask + 1] * 0.7152 + maskBytes[maskY *
      strideMask + maskX * pixelSizeMask + 2] * 0.0722) / 255.0;
00091
00092                                 if (mask.HasAlpha)
00093                                 {
00094                                     weight *= maskBytes[maskY * strideMask + maskX * pixelSizeMask +
      3] / 255.0;
00095                                 }
00096                             }
00097
00098                             if (weight > 0)
00099                             {
00100                                 output[y * strideOutput + x * 4] = input[y * strideInput + x * 4];
00101                                 output[y * strideOutput + x * 4 + 1] = input[y * strideInput + x * 4 +
      1];
00102                                 output[y * strideOutput + x * 4 + 2] = input[y * strideInput + x * 4 +
      2];
00103                                 output[y * strideOutput + x * 4 + 3] = (byte)Math.Round(input[y *
      strideInput + x * 4 + 3] * weight);
00104                             }
00105                             else
00106                             {
00107                                 output[y * strideOutput + x * 4] = 0;
00108                                 output[y * strideOutput + x * 4 + 1] = 0;
00109                                 output[y * strideOutput + x * 4 + 2] = 0;
00110                                 output[y * strideOutput + x * 4 + 3] = 0;
00111                             }
00112                         }
00113                     };
00114                 }
00115                 else
00116                 {
00117                     yLoop = (y) =>
00118                     {
00119                         for (int x = 0; x < width; x++)
00120                         {
00121                             int maskX = (int)Math.Round((bounds.Location.X + (x + 0.5) *
      bounds.Size.Width / width - CachedBounds.Location.X) / CachedBounds.Size.Width * maskWidth);
00122                             int maskY = (int)Math.Round((bounds.Location.Y + (y + 0.5) *
      bounds.Size.Height / height - CachedBounds.Location.Y) / CachedBounds.Size.Height * maskHeight);
00123
00124                             double weight = 0;
00125
00126                             if (maskX >= 0 && maskX < maskWidth && maskY >= 0 && maskY < maskHeight)
00127                             {
00128                                 weight = (maskBytes[maskY * strideMask + maskX * pixelSizeMask] *
      0.2126 + maskBytes[maskY * strideMask + maskX * pixelSizeMask + 1] * 0.7152 + maskBytes[maskY *
      strideMask + maskX * pixelSizeMask + 2] * 0.0722);
00129
00130                                 if (mask.HasAlpha)
00131                                 {
00132                                     weight *= maskBytes[maskY * strideMask + maskX * pixelSizeMask +
      3] / 255.0;
00133                                 }
00134                             }
00135
00136                             if (weight > 0)
00137                             {
00138                                 output[y * strideOutput + x * 4] = input[y * strideInput + x * 3];
00139                                 output[y * strideOutput + x * 4 + 1] = input[y * strideInput + x * 3 +
      1];
00140                                 output[y * strideOutput + x * 4 + 2] = input[y * strideInput + x * 3 +
      2];
00141                                 output[y * strideOutput + x * 4 + 3] = (byte)Math.Round(weight);
00142                             }
00143                             else
00144                             {
00145                                 output[y * strideOutput + x * 4] = 0;
00146                                 output[y * strideOutput + x * 4 + 1] = 0;
```

```
00147                                   output[y * strideOutput + x * 4 + 2] = 0;
00148                                   output[y * strideOutput + x * 4 + 3] = 0;
00149                               }
00150                           }
00151                       };
00152                   }
00153
00154                   if (threads == 1)
00155                   {
00156                       for (int y = 0; y < height; y++)
00157                       {
00158                           yLoop(y);
00159                       }
00160                   }
00161                   else
00162                   {
00163                       ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00164
00165                       Parallel.For(0, height, options, yLoop);
00166                   }
00167
00168               }
00169
00170               DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00171
00172               return new RasterImage(ref disp, width, height, image.HasAlpha, image.Interpolate);
00173
00174           }
00175       }
00176 }
```

8.62 Font.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents a typeface with a specific size.
00026 /// </summary>
00027     public class Font
00028     {
00029 /// <summary>
00030 /// Determines whether text kerning is enabled.  Note that, even when this is set to <see
      langword="false" />, text kerning will be applied on some platforms.  For the best consistency, leave
      this set to <see langword="true"/>.
00031 /// </summary>
00032         public static bool EnableKerning = true;
00033
00034 /// <summary>
00035 /// Represents options to underline text.
00036 /// </summary>
00037         public class FontUnderline
00038         {
00039 /// <summary>
00040 /// Determines whether the underline skips the parts of the glyph that would intersect with it or not.
00041 /// </summary>
00042             public bool SkipDescenders { get; set; }
00043
00044 /// <summary>
00045 /// Determines the position of the top of the underline with respect to the text baseline.  Positive
      values are below the baseline, negative values are above it.  This is expressed as a fraction of the
      font size.
```

```
00046 /// </summary>
00047             public double Position { get; set; }
00048 
00049 /// <summary>
00050 /// Determines the thickness of the underline, expressed as a fraction of the font size.
00051 /// </summary>
00052             public double Thickness { get; set; }
00053 
00054 /// <summary>
00055 /// Determines the caps at the start and end of the underline.
00056 /// </summary>
00057             public LineCaps LineCap { get; set; }
00058 
00059 /// <summary>
00060 /// Determine whether the shape of the underline is slanted to follow the angle of italic fonts.
00061 /// </summary>
00062             public bool FollowItalicAngle { get; set; }
00063 
00064 /// <summary>
00065 /// Create a new underline with the default settings for the specified font family.
00066 /// </summary>
00067 /// <param name="family">The font family to which the underline will be applied.</param>
00068             internal FontUnderline(FontFamily family)
00069             {
00070                 this.SkipDescenders = true;
00071                 this.LineCap = LineCaps.Butt;
00072                 this.FollowItalicAngle = true;
00073 
00074                 if (family.TrueTypeFile != null)
00075                 {
00076                     this.Position = -family.TrueTypeFile.Get1000EmUnderlinePosition() / 1000.0;
00077                     this.Thickness = family.TrueTypeFile.Get1000EmUnderlineThickness() / 1000.0;
00078                 }
00079                 else
00080                 {
00081                     this.Position = 0.3;
00082                     this.Thickness = family.IsBold ?  0.15 :  0.075;
00083                 }
00084 
00085                 if (double.IsNaN(this.Position))
00086                 {
00087                     this.Position = 0.3;
00088                 }
00089 
00090                 if (double.IsNaN(this.Thickness))
00091                 {
00092                     this.Thickness = family.IsBold ?  0.15 :  0.075;
00093                 }
00094             }
00095         }
00096 
00097 /// <summary>
00098 /// Represents detailed information about the metrics of a text string when drawn with a certain font.
00099 /// </summary>
00100         public class DetailedFontMetrics
00101         {
00102 /// <summary>
00103 /// Width of the text (measured on the actual glyph outlines).
00104 /// </summary>
00105             public double Width { get; }
00106 
00107 /// <summary>
00108 /// Height of the text (measured on the actual glyph outlines).
00109 /// </summary>
00110             public double Height { get; }
00111 
00112 /// <summary>
00113 /// How much the leftmost glyph in the string overhangs the glyph origin on the left.  Positive for
      glyphs that hang past the origin (e.g.  italic 'f').
00114 /// </summary>
00115             public double LeftSideBearing { get; }
00116 
00117 /// <summary>
00118 /// How much the rightmost glyph in the string overhangs the glyph end on the right.  Positive for
      glyphs that hang past the end (e.g.  italic 'f').
00119 /// </summary>
00120             public double RightSideBearing { get; }
00121 
00122 /// <summary>
00123 /// Height of the tallest glyph in the string over the baseline.  Always &gt;= 0.
00124 /// </summary>
00125             public double Top { get; }
00126 
00127 /// <summary>
00128 /// Depth of the deepest glyph in the string below the baseline.  Always &lt;= 0.
00129 /// </summary>
00130             public double Bottom { get; }
```

```
00131
00132 /// <summary>
00133 /// Advance width of the text (excluding any left- or right- side bearing).
00134 /// </summary>
00135             public double AdvanceWidth { get; }
00136
00137             internal DetailedFontMetrics(double width, double height, double leftSideBearing, double
      rightSideBearing, double top, double bottom, double advanceWidth)
00138             {
00139                 this.Width = width;
00140                 this.Height = height;
00141                 this.LeftSideBearing = leftSideBearing;
00142                 this.RightSideBearing = rightSideBearing;
00143                 this.Top = top;
00144                 this.Bottom = bottom;
00145                 this.AdvanceWidth = advanceWidth;
00146             }
00147         }
00148
00149 /// <summary>
00150 /// Font size, in graphics units.
00151 /// </summary>
00152         public double FontSize { get; }
00153
00154 /// <summary>
00155 /// Font typeface.
00156 /// </summary>
00157         public FontFamily FontFamily { get; }
00158
00159 /// <summary>
00160 /// Create a new Font object, given the base typeface and the font size.
00161 /// </summary>
00162 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00163 /// <param name="fontSize">The font size, in graphics units.</param>
00164         public Font(FontFamily fontFamily, double fontSize)
00165         {
00166             this.FontFamily = fontFamily;
00167             this.FontSize = fontSize;
00168         }
00169
00170 /// <summary>
00171 /// Create a new Font object, given the base typeface, the font size, and a boolean value determining
      whether text using this font should be underlined.
00172 /// </summary>
00173 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00174 /// <param name="fontSize">The font size, in graphics units.</param>
00175 /// <param name="underlined">A boolean value determining whether text drawn using this font should be
      underlined.  The appearance of the underline can be tweaked by changing the properties of the <see
      cref="Underline"/> property after the font has been created.</param>
00176         public Font(FontFamily fontFamily, double fontSize, bool underlined)
00177         {
00178             this.FontFamily = fontFamily;
00179             this.FontSize = fontSize;
00180
00181             if (underlined)
00182             {
00183                 this.Underline = new FontUnderline(fontFamily);
00184             }
00185         }
00186
00187 /// <summary>
00188 /// Create a new Font object, given the base typeface, the font size, and an object describing the
      underline properties of text drawn using this font.
00189 /// </summary>
00190 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00191 /// <param name="fontSize">The font size, in graphics units.</param>
00192 /// <param name="underline">A <see cref="FontUnderline"/> object describing the underline properties
      of text drawn using this font.</param>
00193         public Font(FontFamily fontFamily, double fontSize, FontUnderline underline)
00194         {
00195             this.FontFamily = fontFamily;
00196             this.FontSize = fontSize;
00197             this.Underline = underline;
00198         }
00199
00200 /// <summary>
00201 /// Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than
      this).  Always &gt;= 0.
00202 /// </summary>
00203         public double Ascent
00204         {
00205             get
00206             {
00207                 if (this.FontFamily.TrueTypeFile == null)
00208                 {
00209                     return 0;
00210                 }
```

```
00211                 else
00212                 {
00213                     return this.FontFamily.TrueTypeFile.Get1000EmAscent() * this.FontSize / 1000;
00214                 }
00215             }
00216         }
00217 
00218 /// <summary>
00219 /// Height above the baseline for a clipping region (Windows ascent).  Always &gt;= 0.
00220 /// </summary>
00221         public double WinAscent
00222         {
00223             get
00224             {
00225                 if (this.FontFamily.TrueTypeFile == null)
00226                 {
00227                     return 0;
00228                 }
00229                 else
00230                 {
00231                     return this.FontFamily.TrueTypeFile.Get1000EmWinAscent() * this.FontSize / 1000;
00232                 }
00233             }
00234         }
00235 
00236 /// <summary>
00237 /// Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than
      this).  Always &lt;= 0.
00238 /// </summary>
00239         public double Descent
00240         {
00241             get
00242             {
00243                 if (this.FontFamily.TrueTypeFile == null)
00244                 {
00245                     return 0;
00246                 }
00247                 else
00248                 {
00249                     return this.FontFamily.TrueTypeFile.Get1000EmDescent() * this.FontSize / 1000;
00250                 }
00251             }
00252         }
00253 
00254 /// <summary>
00255 /// Absolute maximum height over the baseline of the glyphs in the font.  Always &gt;= 0.
00256 /// </summary>
00257         public double YMax
00258         {
00259             get
00260             {
00261                 if (this.FontFamily.TrueTypeFile == null)
00262                 {
00263                     return 0;
00264                 }
00265                 else
00266                 {
00267                     return this.FontFamily.TrueTypeFile.Get1000EmYMax() * this.FontSize / 1000;
00268                 }
00269             }
00270         }
00271 
00272 /// <summary>
00273 /// Absolute maximum depth below the baseline of the glyphs in the font.  Always &lt;= 0.
00274 /// </summary>
00275         public double YMin
00276         {
00277             get
00278             {
00279                 if (this.FontFamily.TrueTypeFile == null)
00280                 {
00281                     return 0;
00282                 }
00283                 else
00284                 {
00285                     return this.FontFamily.TrueTypeFile.Get1000EmYMin() * this.FontSize / 1000;
00286                 }
00287             }
00288         }
00289 
00290 /// <summary>
00291 /// Determines the underline style of text drawn using this font.  If this is <see langword="null"/>,
      the text is not underlined.
00292 /// </summary>
00293         public FontUnderline Underline { get; }
00294 
00295 /// <summary>
```

```
00296 /// Measure the size of a text string when typeset with this font.
00297 /// </summary>
00298 /// <param name="text">The string to measure.</param>
00299 /// <returns>A <see cref="Size"/> object representing the width and height of the text.</returns>
00300         public Size MeasureText(string text)
00301         {
00302             if (this.FontFamily.TrueTypeFile != null && !string.IsNullOrEmpty(text))
00303             {
00304                 double width = 0;
00305                 double yMin = 0;
00306                 double yMax = 0;
00307 
00308                 Point currentGlyphPlacementDelta = new Point();
00309                 Point currentGlyphAdvanceDelta = new Point();
00310                 Point nextGlyphPlacementDelta = new Point();
00311                 Point nextGlyphAdvanceDelta = new Point();
00312 
00313                 for (int i = 0; i < text.Length; i++)
00314                 {
00315                     if (Font.EnableKerning && i < text.Length - 1)
00316                     {
00317                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00318                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00319                         nextGlyphAdvanceDelta = new Point();
00320                         nextGlyphPlacementDelta = new Point();
00321 
00322                         TrueTypeFile.PairKerning kerning =
    this.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00323 
00324                         if (kerning != null)
00325                         {
00326                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
    kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00327                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
    kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00328 
00329                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
    kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00330                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00331                         }
00332                     }
00333 
00334                     width += (this.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) +
    currentGlyphAdvanceDelta.X) * this.FontSize / 1000;
00335                     TrueTypeFile.VerticalMetrics vMet =
    this.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(text[i]);
00336 
00337                     yMin = Math.Min(yMin, (vMet.YMin + currentGlyphPlacementDelta.Y) * this.FontSize /
    1000);
00338                     yMax = Math.Max(yMax, (vMet.YMax + currentGlyphPlacementDelta.Y) * this.FontSize /
    1000);
00339                 }
00340 
00341                 width -= this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[0]).LeftSideBearing
    * this.FontSize / 1000;
00342                 width -= this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[text.Length -
    1]).RightSideBearing * this.FontSize / 1000;
00343 
00344                 return new Size(width, yMax - yMin);
00345             }
00346             else
00347             {
00348                 return new Size(0, 0);
00349             }
00350         }
00351 
00352 /// <summary>
00353 /// Measure all the metrics of a text string when typeset with this font.
00354 /// </summary>
00355 /// <param name="text">The string to measure.</param>
00356 /// <returns>A <see cref="DetailedFontMetrics"/> object representing the metrics of the
    text.</returns>
00357         public DetailedFontMetrics MeasureTextAdvanced(string text)
00358         {
00359             if (this.FontFamily.TrueTypeFile != null && !string.IsNullOrEmpty(text))
00360             {
00361                 double width = 0;
00362                 double yMin = 0;
00363                 double yMax = 0;
00364 
00365                 Point currentGlyphPlacementDelta = new Point();
00366                 Point currentGlyphAdvanceDelta = new Point();
00367                 Point nextGlyphPlacementDelta = new Point();
00368                 Point nextGlyphAdvanceDelta = new Point();
00369 
00370                 for (int i = 0; i < text.Length; i++)
```

```
00371                 {
00372                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00373                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00374 
00375                     if (Font.EnableKerning && i < text.Length - 1)
00376                     {
00377                         nextGlyphAdvanceDelta = new Point();
00378                         nextGlyphPlacementDelta = new Point();
00379 
00380                         TrueTypeFile.PairKerning kerning =
      this.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00381 
00382                         if (kerning != null)
00383                         {
00384                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00385                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00386 
00387                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00388                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00389                         }
00390                     }
00391 
00392                     width += (this.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) +
      currentGlyphAdvanceDelta.X) * this.FontSize / 1000;
00393                     TrueTypeFile.VerticalMetrics vMet =
      this.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(text[i]);
00394 
00395                     yMin = Math.Min(yMin, (vMet.YMin + currentGlyphPlacementDelta.Y) * this.FontSize /
      1000);
00396                     yMax = Math.Max(yMax, (vMet.YMax + currentGlyphPlacementDelta.Y) * this.FontSize /
      1000);
00397                 }
00398 
00399                 double lsb =
      this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[0]).LeftSideBearing * this.FontSize / 1000;
00400                 double rsb = this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[text.Length -
      1]).RightSideBearing * this.FontSize / 1000;
00401 
00402                 double advanceWidth = width;
00403 
00404                 width -= lsb;
00405                 width -= rsb;
00406 
00407                 return new DetailedFontMetrics(width, yMax - yMin, lsb, rsb, yMax, yMin,
      advanceWidth);
00408             }
00409             else
00410             {
00411                 return new DetailedFontMetrics(0, 0, 0, 0, 0, 0, 0);
00412             }
00413         }
00414     }
00415 
00416 
00417 /// <summary>
00418 /// Represents a typeface.
00419 /// </summary>
00420     public class FontFamily
00421     {
00422 /// <summary>
00423 /// The default font library used to resolve font family names.
00424 /// </summary>
00425         public static IFontLibrary DefaultFontLibrary { get; set; } = new DefaultFontLibrary();
00426 
00427 /// <summary>
00428 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, an exception might be raised.  Equivalent to
      DefaultFontLibrary.ResolveFontFamily.
00429 /// </summary>
00430 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00431 /// <returns>If the font family name or the true type file is valid, a <see cref="FontFamily"/> object
      corresponding to the specified font family.</returns>
00432         public static FontFamily ResolveFontFamily(string fontFamily) =>
      DefaultFontLibrary.ResolveFontFamily(fontFamily);
00433 
00434 
00435 /// <summary>
00436 /// Create a new font family from the specified standard font family name.  Equivalent to
      DefaultFontLibrary.ResolveFontFamily.
00437 /// </summary>
00438 /// <param name="standardFontFamily">The standard name of the font family.</param>
00439 /// <returns>A <see cref="FontFamily"/> object corresponding to the specified font family.</returns>
00440         public static FontFamily ResolveFontFamily(StandardFontFamilies standardFontFamily) =>
```

```
    DefaultFontLibrary.ResolveFontFamily(standardFontFamily);
00441
00442 /// <summary>
00443 /// Create a new font family from the specified family name or true type file.  If the family name or
    the true type file are not valid, try to instantiate the font family using
00444 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
    valid, an exception might be raised.  Equivalent to DefaultFontLibrary.ResolveFontFamily.
00445 /// </summary>
00446 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00447 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
    first <paramref name="fontFamily"/> is not valid.</param>
00448 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
    families that is valid.</returns>
00449         public static FontFamily ResolveFontFamily(string fontFamily, params string[] fallback) =>
    DefaultFontLibrary.ResolveFontFamily(fontFamily, fallback);
00450
00451 /// <summary>
00452 /// Create a new font family from the specified family name or true type file.  If the family name or
    the true type file are not valid, try to instantiate the font family using
00453 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
    valid, instantiate a standard font family using the <paramref name="finalFallback"/>.  Equivalent to
    DefaultFontLibrary.ResolveFontFamily.
00454 /// </summary>
00455 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00456 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
    first <paramref name="fontFamily"/> is not valid.</param>
00457 /// <param name="finalFallback">The standard name of the font family that will be used if none of the
    fallback families are valid.</param>
00458 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
    families that is valid.</returns>
00459         public static FontFamily ResolveFontFamily(string fontFamily, StandardFontFamilies
    finalFallback, params string[] fallback) => DefaultFontLibrary.ResolveFontFamily(fontFamily,
    finalFallback, fallback);
00460
00461         internal static object fontFamilyLock = new object();
00462         internal static readonly Dictionary<string, Stream> manifestResources = new Dictionary<string,
    Stream>();
00463
00464         internal static Stream GetManifestResourceStream(string name)
00465         {
00466             if (!manifestResources.ContainsKey(name))
00467             {
00468                 manifestResources.Add(name,
    typeof(FontFamily).Assembly.GetManifestResourceStream(name));
00469             }
00470
00471             return manifestResources[name];
00472         }
00473
00474
00475 /// <summary>
00476 /// The names of the 14 standard families that are guaranteed to be displayed correctly.
00477 /// </summary>
00478         public static string[] StandardFamilies = new string[] { "Times-Roman", "Times-Bold",
    "Times-Italic", "Times-BoldItalic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique",
    "Helvetica-BoldOblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique",
    "Symbol", "ZapfDingbats" };
00479
00480 /// <summary>
00481 /// The names of the resource streams pointing to the included TrueType font files for each of the
    standard 14 font families.
00482 /// </summary>
00483         public static string[] StandardFontFamilyResources = new string[]
00484         {
00485             "VectSharp.StandardFonts.Tinos-Regular.ttf", "VectSharp.StandardFonts.Tinos-Bold.ttf",
    "VectSharp.StandardFonts.Tinos-Italic.ttf", "VectSharp.StandardFonts.Tinos-BoldItalic.ttf",
00486             "VectSharp.StandardFonts.Arimo-Regular.ttf", "VectSharp.StandardFonts.Arimo-Bold.ttf",
    "VectSharp.StandardFonts.Arimo-Italic.ttf", "VectSharp.StandardFonts.Arimo-BoldItalic.ttf",
00487             "VectSharp.StandardFonts.Cousine-Regular.ttf", "VectSharp.StandardFonts.Cousine-Bold.ttf",
    "VectSharp.StandardFonts.Cousine-Italic.ttf", "VectSharp.StandardFonts.Cousine-BoldItalic.ttf",
00488             "VectSharp.StandardFonts.SymbolNeu_GB.ttf",
    "VectSharp.StandardFonts.Levibats-Regular_GB.ttf"
00489         };
00490
00491 /// <summary>
00492 /// Whether this is one of the 14 standard font families or not.
00493 /// </summary>
00494         public bool IsStandardFamily { get; internal set; }
00495
00496 /// <summary>
00497 /// The 14 standard font families.
00498 /// </summary>
00499         public enum StandardFontFamilies
00500         {
00501 /// <summary>
00502 /// Serif normal regular face.
00503 /// </summary>
```

```
00504             TimesRoman,
00505
00506 /// <summary>
00507 /// Serif bold regular face.
00508 /// </summary>
00509             TimesBold,
00510
00511 /// <summary>
00512 /// Serif normal italic face.
00513 /// </summary>
00514             TimesItalic,
00515
00516 /// <summary>
00517 /// Serif bold italic face.
00518 /// </summary>
00519             TimesBoldItalic,
00520
00521 /// <summary>
00522 /// Sans-serif normal regular face.
00523 /// </summary>
00524             Helvetica,
00525
00526 /// <summary>
00527 /// Sans-serif bold regular face.
00528 /// </summary>
00529             HelveticaBold,
00530
00531 /// <summary>
00532 /// Sans-serif normal oblique face.
00533 /// </summary>
00534             HelveticaOblique,
00535
00536 /// <summary>
00537 /// Sans-serif bold oblique face.
00538 /// </summary>
00539             HelveticaBoldOblique,
00540
00541 /// <summary>
00542 /// Monospace normal regular face.
00543 /// </summary>
00544             Courier,
00545
00546 /// <summary>
00547 /// Monospace bold regular face.
00548 /// </summary>
00549             CourierBold,
00550
00551 /// <summary>
00552 /// Monospace normal oblique face.
00553 /// </summary>
00554             CourierOblique,
00555
00556 /// <summary>
00557 /// Monospace bold oblique face.
00558 /// </summary>
00559             CourierBoldOblique,
00560
00561 /// <summary>
00562 /// Symbol font.
00563 /// </summary>
00564             Symbol,
00565
00566 /// <summary>
00567 /// Dingbat font.
00568 /// </summary>
00569             ZapfDingbats
00570         }
00571
00572 /// <summary>
00573 /// Full path to the TrueType font file for this font family (or, if this is a standard font family,
      name of the font family).
00574 /// </summary>
00575         public string FileName { get; internal set; }
00576
00577 /// <summary>
00578 /// Name of the font family, including any variantes.
00579 /// </summary>
00580         public string FamilyName { get; internal set; }
00581
00582 /// <summary>
00583 /// Parsed TrueType font file for this font family.
00584 /// See also:  <seealso cref="VectSharp.TrueTypeFile"/>.
00585 /// </summary>
00586         public TrueTypeFile TrueTypeFile { get; }
00587
00588 /// <summary>
00589 /// Whether this font is bold or not.  This is set based on the information included in the OS/2 table
```

```
    of the TrueType file.
00590 /// </summary>
00591         public bool IsBold { get; internal set; }
00592
00593 /// <summary>
00594 /// Whether this font is italic or oblique or not.  This is set based on the information included in
    the OS/2 table of the TrueType file.
00595 /// </summary>
00596         public bool IsItalic { get; internal set; }
00597
00598 /// <summary>
00599 /// Whether this font is oblique or not.  This is set based on the information included in the OS/2
    table of the TrueType file.
00600 /// </summary>
00601         public bool IsOblique { get; internal set; }
00602
00603 /// <summary>
00604 /// Create a new <see cref="FontFamily"/>.
00605 /// </summary>
00606 /// <param name="fileName">The full path to the TrueType font file for this font family or the name of
    a standard font family.</param>
00607         [Obsolete("Please use the FontFamily.ResolveFontFamily(string) method instead!", true)]
00608         public FontFamily(string fileName)
00609         {
00610             lock (fontFamilyLock)
00611             {
00612                 FontFamily resolved = DefaultFontLibrary.ResolveFontFamily(fileName);
00613
00614                 this.FileName = resolved.FileName;
00615                 this.FamilyName = resolved.FamilyName;
00616                 this.TrueTypeFile = resolved.TrueTypeFile;
00617                 this.IsOblique = resolved.IsOblique;
00618                 this.IsBold = resolved.IsBold;
00619                 this.IsItalic = resolved.IsItalic;
00620                 this.IsStandardFamily = resolved.IsStandardFamily;
00621             }
00622         }
00623
00624         internal FontFamily()
00625         {
00626
00627         }
00628
00629 /// <summary>
00630 /// Create a new <see cref="FontFamily"/>.
00631 /// </summary>
00632 /// <param name="ttfStream">A stream containing a file in TTF format.</param>
00633         public FontFamily(Stream ttfStream)
00634         {
00635             lock (fontFamilyLock)
00636             {
00637                 IsStandardFamily = false;
00638
00639                 TrueTypeFile = TrueTypeFile.CreateTrueTypeFile(ttfStream);
00640
00641                 FileName = TrueTypeFile.GetFontFamilyName();
00642                 FamilyName = TrueTypeFile.GetFullFontFamilyName() ??  FileName;
00643                 this.IsBold = TrueTypeFile.IsBold();
00644                 this.IsItalic = TrueTypeFile.IsItalic();
00645                 this.IsOblique = TrueTypeFile.IsOblique();
00646             }
00647         }
00648
00649 /// <summary>
00650 /// Create a new <see cref="FontFamily"/>.
00651 /// </summary>
00652 /// <param name="ttf">A font file in TTF format.</param>
00653         public FontFamily(TrueTypeFile ttf)
00654         {
00655             lock (fontFamilyLock)
00656             {
00657                 IsStandardFamily = false;
00658
00659                 TrueTypeFile = ttf;
00660
00661                 FileName = TrueTypeFile.GetFontFamilyName();
00662                 FamilyName = TrueTypeFile.GetFullFontFamilyName() ??  FileName;
00663                 this.IsBold = TrueTypeFile.IsBold();
00664                 this.IsItalic = TrueTypeFile.IsItalic();
00665                 this.IsOblique = TrueTypeFile.IsOblique();
00666             }
00667         }
00668
00669 /// <summary>
00670 /// Create a new standard <see cref="FontFamily"/>.
00671 /// </summary>
00672 /// <param name="standardFontFamily">The standard font family.</param>
```

```
00673         [Obsolete("Please use the FontFamily.ResolveFontFamily(StandardFontFamilies) method instead!",
      true)]
00674         public FontFamily(StandardFontFamilies standardFontFamily)
00675         {
00676             lock (fontFamilyLock)
00677             {
00678                 FontFamily resolved = DefaultFontLibrary.ResolveFontFamily(standardFontFamily);
00679 
00680                 this.FileName = resolved.FileName;
00681                 this.FamilyName = resolved.FamilyName;
00682                 this.TrueTypeFile = resolved.TrueTypeFile;
00683                 this.IsOblique = resolved.IsOblique;
00684                 this.IsBold = resolved.IsBold;
00685                 this.IsItalic = resolved.IsItalic;
00686                 this.IsStandardFamily = resolved.IsStandardFamily;
00687             }
00688         }
00689     }
00690 
00691 /// <summary>
00692 /// Represents a FontFamily created from a resource stream.
00693 /// </summary>
00694     public class ResourceFontFamily :  FontFamily
00695     {
00696 /// <summary>
00697 /// The name of the embedded resource, which will be parsed using
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)</code>.
00698 /// </summary>
00699         public string ResourceName;
00700 
00701 /// <summary>
00702 /// Create a new <see cref="ResourceFontFamily"/> from the specified <paramref name="resourceStream"/>
      containing a TTF file, passing the specified <paramref name="resourceName"/> to the
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)"</code> method.
00703 /// </summary>
00704 /// <param name="resourceStream">A resource stream containing a TTF file.</param>
00705 /// <param name="resourceName">The name of the embedded resource, which will be parsed using
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)</code>.</param>
00706         public ResourceFontFamily(System.IO.Stream resourceStream, string resourceName) :
      base(resourceStream)
00707         {
00708             this.ResourceName = resourceName;
00709         }
00710     }
00711 
00712 }
```

8.63 FontLibrary.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.IO;
00022 
00023 namespace VectSharp
00024 {
00025 /// <summary>
00026 /// Represents a font library with methods to create <see cref="FontFamily"/> objects from a string or
      from <see cref="FontFamily.StandardFontFamilies"/>.
00027 /// </summary>
00028     public interface IFontLibrary
00029     {
00030 /// <summary>
00031 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, an exception might be raised.
00032 /// </summary>
```

```
00033 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00034 /// <returns>If the font family name or the true type file is valid, a <see cref="FontFamily"/> object
      corresponding to the specified font family.</returns>
00035         FontFamily ResolveFontFamily(string fontFamily);
00036 
00037 /// <summary>
00038 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00039 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, an exception might be raised.
00040 /// </summary>
00041 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00042 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00043 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00044         FontFamily ResolveFontFamily(string fontFamily, params string[] fallback);
00045 
00046 /// <summary>
00047 /// Create a new font family from the specified standard font family name.
00048 /// </summary>
00049 /// <param name="standardFontFamily">The standard name of the font family.</param>
00050 /// <returns>A <see cref="FontFamily"/> object corresponding to the specified font family.</returns>
00051         FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies standardFontFamily);
00052 
00053 /// <summary>
00054 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00055 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, instantiate a standard font family using the <paramref name="finalFallback"/>.
00056 /// </summary>
00057 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00058 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00059 /// <param name="finalFallback">The standard name of the font family that will be used if none of the
      fallback families are valid.</param>
00060 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00061         FontFamily ResolveFontFamily(string fontFamily, FontFamily.StandardFontFamilies finalFallback,
      params string[] fallback);
00062     }
00063 
00064 /// <summary>
00065 /// Abstract class with a default implementation of font family fallbacks.
00066 /// </summary>
00067     public abstract class FontLibrary :  IFontLibrary
00068     {
00069 /// <inheritdoc/>
00070         public abstract FontFamily ResolveFontFamily(string fontFamily);
00071 
00072 /// <inheritdoc/>
00073         public abstract FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily);
00074 
00075 /// <inheritdoc/>
00076         public virtual FontFamily ResolveFontFamily(string fontFamily, params string[] fallback)
00077         {
00078             bool found = false;
00079 
00080             FontFamily tbr = null;
00081 
00082             try
00083             {
00084                 tbr = ResolveFontFamily(fontFamily);
00085                 if (tbr.TrueTypeFile != null)
00086                 {
00087                     found = true;
00088                 }
00089             }
00090             catch
00091             {
00092                 tbr = null;
00093                 found = false;
00094             }
00095 
00096             if (!found)
00097             {
00098                 for (int i = 0; i < fallback.Length; i++)
00099                 {
00100                     try
00101                     {
00102                         tbr = ResolveFontFamily(fontFamily);
00103                         if (tbr.TrueTypeFile != null)
00104                         {
00105                             found = true;
00106                         }
00107                     }
```

```
00108                     catch
00109                     {
00110                         tbr = null;
00111                         found = false;
00112                     }
00113 
00114                     if (found)
00115                     {
00116                         return tbr;
00117                     }
00118                 }
00119             }
00120             else
00121             {
00122                 return tbr;
00123             }
00124 
00125             throw new FontFamilyCreationException(fontFamily);
00126         }
00127 
00128 /// <inheritdoc/>
00129         public virtual FontFamily ResolveFontFamily(string fontFamily, FontFamily.StandardFontFamilies
    finalFallback, params string[] fallback)
00130         {
00131             bool found = false;
00132 
00133             FontFamily tbr = null;
00134 
00135             try
00136             {
00137                 tbr = ResolveFontFamily(fontFamily);
00138                 if (tbr.TrueTypeFile != null)
00139                 {
00140                     found = true;
00141                 }
00142             }
00143             catch
00144             {
00145                 tbr = null;
00146                 found = false;
00147             }
00148 
00149             if (!found)
00150             {
00151                 for (int i = 0; i < fallback.Length; i++)
00152                 {
00153                     try
00154                     {
00155                         tbr = ResolveFontFamily(fontFamily);
00156                         if (tbr.TrueTypeFile != null)
00157                         {
00158                             found = true;
00159                         }
00160                     }
00161                     catch
00162                     {
00163                         tbr = null;
00164                         found = false;
00165                     }
00166 
00167                     if (found)
00168                     {
00169                         return tbr;
00170                     }
00171                 }
00172             }
00173             else
00174             {
00175                 return tbr;
00176             }
00177 
00178             return ResolveFontFamily(finalFallback);
00179         }
00180     }
00181 
00182 /// <summary>
00183 /// A font library that can be used to cache and resolve font family names.
00184 /// </summary>
00185     public class SimpleFontLibrary :  FontLibrary
00186     {
00187         private Dictionary<string, string> KnownFonts = new Dictionary<string, string>();
00188         private Dictionary<string, string> NotLoadedFonts = new Dictionary<string, string>();
00189         private Dictionary<string, FontFamily> LoadedFonts = new Dictionary<string, FontFamily>();
00190         private Dictionary<FontFamily.StandardFontFamilies, FontFamily> Fallbacks = new
    Dictionary<FontFamily.StandardFontFamilies, FontFamily>();
00191         private DefaultFontLibrary defaultLibrary = new DefaultFontLibrary();
00192 
```

```
00193 /// <summary>
00194 /// Create a new <see cref="SimpleFontLibrary"/> instance.
00195 /// </summary>
00196 /// <param name="standardFontLibrary">An existing font library that will be used to resolve the
      standard font families.</param>
00197         public SimpleFontLibrary(IFontLibrary standardFontLibrary)
00198         {
00199             for (int i = 0; i < 14; i++)
00200             {
00201                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00202                 FontFamily resolved = standardFontLibrary.ResolveFontFamily(stdFF);
00203 
00204                 Fallbacks[stdFF] = resolved;
00205                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00206                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00207                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00208                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
      FontFamily.StandardFamilies[i];
00209                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
      FontFamily.StandardFamilies[i];
00210                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00211             }
00212         }
00213 
00214 /// <summary>
00215 /// Create a new <see cref="SimpleFontLibrary"/> instance, using the default font library to resolve
      the standard font families.
00216 /// </summary>
00217         public SimpleFontLibrary() :  this(FontFamily.DefaultFontLibrary)
00218         {
00219 
00220         }
00221 
00222 /// <summary>
00223 /// Create a new <see cref="SimpleFontLibrary"/> instance, with the specified replacements for the
      standard font families.
00224 /// </summary>
00225 /// <param name="timesRoman">The font family to use for the Times-Roman standard font.</param>
00226 /// <param name="timesBold">The font family to use for the Times-Bold standard font.</param>
00227 /// <param name="timesItalic">The font family to use for the Times-Italic standard font.</param>
00228 /// <param name="timesBoldItalic">The font family to use for the Times-BoldItalic standard
      font.</param>
00229 /// <param name="helvetica">The font family to use for the Helvetica standard font.</param>
00230 /// <param name="helveticaBold">The font family to use for the Helvetica-Bold standard font.</param>
00231 /// <param name="helveticaOblique">The font family to use for the Helvetica-Oblique standard
      font.</param>
00232 /// <param name="helveticaBoldOblique">The font family to use for the Helvetica-BoldOblique standard
      font.</param>
00233 /// <param name="courier">The font family to use for the Courier standard font.</param>
00234 /// <param name="courierBold">The font family to use for the Courier-Bold standard font.</param>
00235 /// <param name="courierOblique">The font family to use for the Courier-Oblique standard font.</param>
00236 /// <param name="courierBoldOblique">The font family to use for the Courier-BoldOblique standard
      font.</param>
00237 /// <param name="symbol">The font family to use for the Symbol standard font.</param>
00238 /// <param name="zapfdingbats">The font family to use for the Zapfdingbats standard font.</param>
00239         public SimpleFontLibrary(FontFamily timesRoman, FontFamily timesBold, FontFamily timesItalic,
      FontFamily timesBoldItalic,
00240             FontFamily helvetica, FontFamily helveticaBold, FontFamily helveticaOblique, FontFamily
      helveticaBoldOblique,
00241             FontFamily courier, FontFamily courierBold, FontFamily courierOblique, FontFamily
      courierBoldOblique,
00242             FontFamily symbol, FontFamily zapfdingbats)
00243         {
00244             Fallbacks[FontFamily.StandardFontFamilies.TimesRoman] = timesRoman;
00245             Fallbacks[FontFamily.StandardFontFamilies.TimesBold] = timesBold;
00246             Fallbacks[FontFamily.StandardFontFamilies.TimesItalic] = timesItalic;
00247             Fallbacks[FontFamily.StandardFontFamilies.TimesBoldItalic] = timesBoldItalic;
00248 
00249             Fallbacks[FontFamily.StandardFontFamilies.Helvetica] = helvetica;
00250             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBold] = helveticaBold;
00251             Fallbacks[FontFamily.StandardFontFamilies.HelveticaOblique] = helveticaOblique;
00252             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBoldOblique] = helveticaBoldOblique;
00253 
00254             Fallbacks[FontFamily.StandardFontFamilies.Courier] = courier;
00255             Fallbacks[FontFamily.StandardFontFamilies.CourierBold] = courierBold;
00256             Fallbacks[FontFamily.StandardFontFamilies.CourierOblique] = courierOblique;
00257             Fallbacks[FontFamily.StandardFontFamilies.CourierBoldOblique] = courierBoldOblique;
00258 
00259             Fallbacks[FontFamily.StandardFontFamilies.Symbol] = symbol;
00260             Fallbacks[FontFamily.StandardFontFamilies.ZapfDingbats] = zapfdingbats;
00261 
00262             for (int i = 0; i < 14; i++)
00263             {
00264                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00265                 FontFamily resolved = Fallbacks[stdFF];
00266                 resolved.IsStandardFamily = true;
00267                 resolved.FileName = FontFamily.StandardFamilies[i];
```

```
00268                 resolved.FamilyName = FontFamily.StandardFamilies[i].Replace("-", " ");
00269
00270                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00271                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00272                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00273                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00274                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00275                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00276             }
00277         }
00278
00279
00280 /// <summary>
00281 /// Create a new <see cref="SimpleFontLibrary"/> instance, with the specified replacements for the
    standard font families.
00282 /// </summary>
00283 /// <param name="timesRoman">The font family to use for the Times-Roman standard font.</param>
00284 /// <param name="timesBold">The font family to use for the Times-Bold standard font.</param>
00285 /// <param name="timesItalic">The font family to use for the Times-Italic standard font.</param>
00286 /// <param name="timesBoldItalic">The font family to use for the Times-BoldItalic standard
    font.</param>
00287 /// <param name="helvetica">The font family to use for the Helvetica standard font.</param>
00288 /// <param name="helveticaBold">The font family to use for the Helvetica-Bold standard font.</param>
00289 /// <param name="helveticaOblique">The font family to use for the Helvetica-Oblique standard
    font.</param>
00290 /// <param name="helveticaBoldOblique">The font family to use for the Helvetica-BoldOblique standard
    font.</param>
00291 /// <param name="courier">The font family to use for the Courier standard font.</param>
00292 /// <param name="courierBold">The font family to use for the Courier-Bold standard font.</param>
00293 /// <param name="courierOblique">The font family to use for the Courier-Oblique standard font.</param>
00294 /// <param name="courierBoldOblique">The font family to use for the Courier-BoldOblique standard
    font.</param>
00295 /// <param name="symbol">The font family to use for the Symbol standard font.</param>
00296 /// <param name="zapfdingbats">The font family to use for the Zapfdingbats standard font.</param>
00297         public SimpleFontLibrary(string timesRoman, string timesBold, string timesItalic, string
    timesBoldItalic,
00298             string helvetica, string helveticaBold, string helveticaOblique, string
    helveticaBoldOblique,
00299             string courier, string courierBold, string courierOblique, string courierBoldOblique,
00300             string symbol, string zapfdingbats)
00301         {
00302             Fallbacks[FontFamily.StandardFontFamilies.TimesRoman] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesRoman);
00303             Fallbacks[FontFamily.StandardFontFamilies.TimesBold] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesBold);
00304             Fallbacks[FontFamily.StandardFontFamilies.TimesItalic] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesItalic);
00305             Fallbacks[FontFamily.StandardFontFamilies.TimesBoldItalic] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesBoldItalic);
00306
00307             Fallbacks[FontFamily.StandardFontFamilies.Helvetica] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helvetica);
00308             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBold] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaBold);
00309             Fallbacks[FontFamily.StandardFontFamilies.HelveticaOblique] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaOblique);
00310             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBoldOblique] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaBoldOblique);
00311
00312             Fallbacks[FontFamily.StandardFontFamilies.Courier] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(courier);
00313             Fallbacks[FontFamily.StandardFontFamilies.CourierBold] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(courierBold);
00314             Fallbacks[FontFamily.StandardFontFamilies.CourierOblique] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(courierOblique);
00315             Fallbacks[FontFamily.StandardFontFamilies.CourierBoldOblique] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(courierBoldOblique);
00316
00317             Fallbacks[FontFamily.StandardFontFamilies.Symbol] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(symbol);
00318             Fallbacks[FontFamily.StandardFontFamilies.ZapfDingbats] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(zapfdingbats);
00319
00320             for (int i = 0; i < 14; i++)
00321             {
00322                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00323                 FontFamily resolved = Fallbacks[stdFF];
00324                 resolved.IsStandardFamily = true;
00325                 resolved.FileName = FontFamily.StandardFamilies[i];
00326                 resolved.FamilyName = FontFamily.StandardFamilies[i].Replace("-", " ");
00327
00328                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00329                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00330                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00331                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
```

```
FontFamily.StandardFamilies[i];
00332                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
FontFamily.StandardFamilies[i];
00333                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00334             }
00335         }
00336
00337
00338 /// <summary>
00339 /// Add the specified font family to the library with the specified name.
00340 /// </summary>
00341 /// <param name="fontFamilyName">The name of the font family.</param>
00342 /// <param name="fontFamily">The font family to add.</param>
00343         public void Add(string fontFamilyName, FontFamily fontFamily)
00344         {
00345             this.LoadedFonts[fontFamilyName] = fontFamily;
00346             this.KnownFonts[fontFamilyName] = fontFamilyName;
00347
00348             if (fontFamily.TrueTypeFile != null)
00349             {
00350                 this.KnownFonts[fontFamily.TrueTypeFile.GetFontFamilyName()] = fontFamilyName;
00351                 this.KnownFonts[fontFamily.TrueTypeFile.GetFullFontFamilyName()] = fontFamilyName;
00352                 this.KnownFonts[fontFamily.TrueTypeFile.GetFontName()] = fontFamilyName;
00353             }
00354         }
00355
00356 /// <summary>
00357 /// Add the specified font family to the library.
00358 /// </summary>
00359 /// <param name="fontFamily">The font family to add.</param>
00360         public void Add(FontFamily fontFamily)
00361         {
00362             if (fontFamily.TrueTypeFile != null)
00363             {
00364                 this.Add(fontFamily.TrueTypeFile.GetFullFontFamilyName(), fontFamily);
00365             }
00366         }
00367
00368 /// <summary>
00369 /// Add the font family contained in the specified True Type Font file to the library.
00370 /// </summary>
00371 /// <param name="fileName">The path to the TTF file containing the font family.</param>
00372         public void Add(string fileName)
00373         {
00374             FontFamily fontFamily = FontFamily.DefaultFontLibrary.ResolveFontFamily(fileName);
00375
00376             if (fontFamily.TrueTypeFile != null)
00377             {
00378                 this.Add(fontFamily.TrueTypeFile.GetFullFontFamilyName(), fontFamily);
00379             }
00380         }
00381
00382 /// <summary>
00383 /// Add the font family contained in the specified True Type Font file to the library, with the
specified name.  The font family is not loaded until it is requested for the first time.
00384 /// </summary>
00385 /// <param name="fontFamily">The name of the font family.</param>
00386 /// <param name="fileName">The path to the TTF file containing the font family.</param>
00387         public void Add(string fontFamily, string fileName)
00388         {
00389             this.KnownFonts[fontFamily] = fontFamily;
00390             this.NotLoadedFonts[fontFamily] = fileName;
00391         }
00392
00393 /// <inheritdoc/>
00394         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
standardFontFamily)
00395         {
00396             return Fallbacks[standardFontFamily];
00397         }
00398
00399 /// <inheritdoc/>
00400         public override FontFamily ResolveFontFamily(string fontFamily)
00401         {
00402             if (KnownFonts.TryGetValue(fontFamily, out string knownFontName))
00403             {
00404                 if (LoadedFonts.TryGetValue(knownFontName, out FontFamily tbr))
00405                 {
00406                     return tbr;
00407                 }
00408                 else
00409                 {
00410                     if (NotLoadedFonts.TryGetValue(knownFontName, out string ttfFile))
00411                     {
00412                         tbr = defaultLibrary.ResolveFontFamily(ttfFile);
00413
00414                         if (tbr.TrueTypeFile != null)
```

```
00415                         {
00416                             string familyName = tbr.TrueTypeFile.GetFontFamilyName();
00417 
00418                             this.LoadedFonts[familyName] = tbr;
00419                             this.KnownFonts[familyName] = familyName;
00420                             this.KnownFonts[tbr.TrueTypeFile.GetFontName()] = familyName;
00421                         }
00422 
00423                         return tbr;
00424                     }
00425                     else
00426                     {
00427                         return defaultLibrary.ResolveFontFamily(fontFamily);
00428                     }
00429                 }
00430             }
00431             else
00432             {
00433                 return defaultLibrary.ResolveFontFamily(fontFamily);
00434             }
00435         }
00436     }
00437 
00438 /// <summary>
00439 /// An exception that occurs while creating a <see cref="FontFamily"/>.
00440 /// </summary>
00441     public class FontFamilyCreationException :  Exception
00442     {
00443 /// <summary>
00444 /// The name of the font family that was being created.
00445 /// </summary>
00446         public string FontFamily { get; }
00447 
00448 /// <summary>
00449 /// Create a new <see cref="FontFamilyCreationException"/> instance.
00450 /// </summary>
00451 /// <param name="fontFamily">The name of the font family that was being created.</param>
00452         public FontFamilyCreationException(string fontFamily) :  base("The font family \"" +
      fontFamily + "\" could not be created!")
00453         {
00454             this.FontFamily = fontFamily;
00455         }
00456     }
00457 
00458 /// <summary>
00459 /// A default font library that resolves standard families using the embedded fonts.
00460 /// </summary>
00461     public class DefaultFontLibrary :  FontLibrary
00462     {
00463 /// <inheritdoc/>
00464         public override FontFamily ResolveFontFamily(string fontFamily)
00465         {
00466             lock (FontFamily.fontFamilyLock)
00467             {
00468                 bool isStandardFamily;
00469 
00470                 if (FontFamily.StandardFamilies.Contains(fontFamily) ||
      FontFamily.StandardFamilies.Contains(fontFamily.Replace(" ", "-")))
00471                 {
00472                     isStandardFamily = true;
00473                 }
00474                 else
00475                 {
00476                     isStandardFamily = false;
00477                 }
00478 
00479                 if (isStandardFamily)
00480                 {
00481                     fontFamily = fontFamily.Replace(" ", "-");
00482                     Stream ttfStream =
      FontFamily.GetManifestResourceStream(FontFamily.StandardFontFamilyResources[Array.IndexOf(FontFamily.StandardFamilies,
      fontFamily)]);
00483 
00484                     FontFamily tbr = new FontFamily(ttfStream);
00485                     tbr.IsStandardFamily = true;
00486                     tbr.FileName = fontFamily;
00487                     tbr.FamilyName = tbr.FileName.Replace("-", " ");
00488 
00489                     if (fontFamily == "Times-Italic" || fontFamily == "Times-BoldItalic" || fontFamily
      == "Helvetica-Oblique" || fontFamily == "Helvetica-BoldOblique" || fontFamily == "Courier-Oblique" ||
      fontFamily == "Courier-BoldOblique")
00490                     {
00491                         tbr.IsItalic = true;
00492                         tbr.IsOblique = (fontFamily == "Courier-Oblique" || fontFamily ==
      "Courier-BoldOblique");
00493                     }
00494                     else
```

```
00495                     {
00496                         tbr.IsItalic = false;
00497                         tbr.IsOblique = false;
00498                     }
00499 
00500                     return tbr;
00501                 }
00502                 else
00503                 {
00504                     try
00505                     {
00506                         FontFamily tbr = new FontFamily(TrueTypeFile.CreateTrueTypeFile(fontFamily));
00507                         tbr.FileName = fontFamily;
00508                         tbr.FamilyName = tbr.TrueTypeFile?.GetFullFontFamilyName() ??  tbr.FileName;
00509                         return tbr;
00510                     }
00511                     catch
00512                     {
00513                         FontFamily tbr = new FontFamily();
00514                         tbr.FileName = fontFamily;
00515                         tbr.FamilyName = fontFamily;
00516                         return tbr;
00517                     }
00518                 }
00519             }
00520         }
00521 
00522 /// <inheritdoc/>
00523         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
    standardFontFamily)
00524         {
00525             lock (FontFamily.fontFamilyLock)
00526             {
00527                 Stream ttfStream =
    FontFamily.GetManifestResourceStream(FontFamily.StandardFontFamilyResources[(int)standardFontFamily]);
00528                 FontFamily tbr = new FontFamily(ttfStream);
00529 
00530 
00531                 tbr.IsStandardFamily = true;
00532 
00533                 tbr.FileName = FontFamily.StandardFamilies[(int)standardFontFamily];
00534                 tbr.FamilyName = tbr.FileName.Replace("-", " ");
00535 
00536                 if (tbr.FileName == "Times-Italic" || tbr.FileName == "Times-BoldItalic" ||
    tbr.FileName == "Helvetica-Oblique" || tbr.FileName == "Helvetica-BoldOblique" || tbr.FileName ==
    "Courier-Oblique" || tbr.FileName == "Courier-BoldOblique")
00537                 {
00538                     tbr.IsItalic = true;
00539                     tbr.IsOblique = (tbr.FileName == "Courier-Oblique" || tbr.FileName ==
    "Courier-BoldOblique");
00540                 }
00541                 else
00542                 {
00543                     tbr.IsItalic = false;
00544                     tbr.IsOblique = false;
00545                 }
00546 
00547                 return tbr;
00548             }
00549         }
00550     }
00551 
00552 /// <summary>
00553 /// A font library that resolves fonts from a folder containing TrueType files.
00554 /// </summary>
00555     public class FolderFontLibrary :  FontLibrary
00556     {
00557         private Dictionary<string, FontFamily> LoadedFonts = new Dictionary<string, FontFamily>();
00558         private Dictionary<string, string> KnownFonts = new Dictionary<string, string>();
00559         private IFontLibrary StandardFontLibrary;
00560 
00561 /// <summary>
00562 /// Creates a new <see cref="FolderFontLibrary"/> using fonts from the specified path, and using the
    <see cref="FontFamily.DefaultFontLibrary"/> to resolve the standard font families.
00563 /// </summary>
00564 /// <param name="folderPath">The path to the folder containing the TrueType files.</param>
00565         public FolderFontLibrary(string folderPath) :  this(folderPath, FontFamily.DefaultFontLibrary)
    { }
00566 
00567 /// <summary>
00568 /// Creates a new <see cref="FolderFontLibrary"/> using fonts from the specified path, and using the
    specified <see cref="IFontLibrary"/> to resolve the standard font families.
00569 /// </summary>
00570 /// <param name="folderPath">The path to the folder containing the TrueType files.</param>
00571 /// <param name="standardFontLibrary">The <see cref="IFontLibrary"/> to use when resolving standard
    font families.</param>
00572         public FolderFontLibrary(string folderPath, IFontLibrary standardFontLibrary)
```

```
00573         {
00574             foreach (string file in Directory.GetFiles(folderPath))
00575             {
00576                 bool isValid = TrueTypeFile.GetNames(file, out List<TrueTypeFile.TrueTypeName> names);
00577 
00578                 if (isValid && names != null)
00579                 {
00580                     foreach (TrueTypeFile.TrueTypeName name in names)
00581                     {
00582                         switch (name.NameId)
00583                         {
00584                             case TrueTypeFile.TrueTypeName.NameIdentifier.FullName:
00585                             case TrueTypeFile.TrueTypeName.NameIdentifier.PostScriptName:
00586                                 KnownFonts[name.Name] = file;
00587                                 break;
00588                             case TrueTypeFile.TrueTypeName.NameIdentifier.FontFamily:
00589                             case TrueTypeFile.TrueTypeName.NameIdentifier.PreferredFamily:
00590                                 if (!KnownFonts.ContainsKey(name.Name))
00591                                 {
00592                                     KnownFonts[name.Name] = file;
00593                                 }
00594                                 break;
00595                         }
00596                     }
00597                 }
00598             }
00599 
00600             StandardFontLibrary = standardFontLibrary;
00601         }
00602 
00603 /// <inheritdoc/>
00604         public override FontFamily ResolveFontFamily(string fontFamily)
00605         {
00606             if (KnownFonts.TryGetValue(fontFamily, out string knownFontName))
00607             {
00608                 if (LoadedFonts.TryGetValue(knownFontName, out FontFamily tbr))
00609                 {
00610                     return tbr;
00611                 }
00612                 else
00613                 {
00614                     tbr = new FontFamily(TrueTypeFile.CreateTrueTypeFile(knownFontName));
00615                     tbr.FileName = knownFontName;
00616                     tbr.FamilyName = tbr.TrueTypeFile?.GetFullFontFamilyName() ??  tbr.FileName;
00617 
00618                     if (tbr.TrueTypeFile != null)
00619                     {
00620                         this.LoadedFonts[knownFontName] = tbr;
00621                     }
00622 
00623                     return tbr;
00624                 }
00625             }
00626             else
00627             {
00628                 FontFamily tbr = new FontFamily();
00629                 tbr.FileName = fontFamily;
00630                 tbr.FamilyName = fontFamily;
00631                 return tbr;
00632             }
00633         }
00634 
00635 /// <inheritdoc/>
00636         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00637         {
00638             FontFamily attempt =
      ResolveFontFamily(FontFamily.StandardFamilies[(int)standardFontFamily]);
00639 
00640             if (attempt != null && attempt.TrueTypeFile != null)
00641             {
00642                 return attempt;
00643             }
00644             else
00645             {
00646                 return StandardFontLibrary.ResolveFontFamily(standardFontFamily);
00647             }
00648         }
00649     }
00650 
00651 /// <summary>
00652 /// A font library that tries to resolve fonts using other font libraries.
00653 /// </summary>
00654     public class MultiFontLibrary :  FontLibrary
00655     {
00656         private FontLibrary[] Libraries;
00657         private DefaultFontLibrary DefaultLibrary = new DefaultFontLibrary();
```

```
00658
00659 /// <summary>
00660 /// Creates a new <see cref="MultiFontLibrary"/> resolving fonts using the specified <paramref
      name="libraries"/>.
00661 /// </summary>
00662 /// <param name="libraries">The font libraries that will be used, in order, to resolve the font
      families.</param>
00663 /// <exception cref="ArgumentException">Thrown if the <paramref name="libraries"/> do not contain any
      element.</exception>
00664         public MultiFontLibrary(params FontLibrary[] libraries) :
      this((IEnumerable<FontLibrary>)libraries) { }
00665
00666
00667 /// <summary>
00668 /// Creates a new <see cref="MultiFontLibrary"/> resolving fonts using the specified <paramref
      name="libraries"/>.
00669 /// </summary>
00670 /// <param name="libraries">The font libraries that will be used, in order, to resolve the font
      families.</param>
00671 /// <exception cref="ArgumentException">Thrown if the <paramref name="libraries"/> do not contain any
      element.</exception>
00672         public MultiFontLibrary(IEnumerable<FontLibrary> libraries)
00673         {
00674             this.Libraries = libraries.ToArray();
00675
00676             if (this.Libraries.Length == 0)
00677             {
00678                 throw new ArgumentException("No font library has been provided!");
00679             }
00680         }
00681
00682 /// <inheritdoc/>
00683         public override FontFamily ResolveFontFamily(string fontFamily)
00684         {
00685             for (int i = 0; i < Libraries.Length; i++)
00686             {
00687                 FontFamily attempt = Libraries[i].ResolveFontFamily(fontFamily);
00688
00689                 if (attempt != null && attempt.TrueTypeFile != null)
00690                 {
00691                     return attempt;
00692                 }
00693             }
00694
00695             return DefaultLibrary.ResolveFontFamily(fontFamily);
00696         }
00697
00698 /// <inheritdoc/>
00699         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00700         {
00701             for (int i = 0; i < Libraries.Length; i++)
00702             {
00703                 FontFamily attempt = Libraries[i].ResolveFontFamily(standardFontFamily);
00704
00705                 if (attempt != null && attempt.TrueTypeFile != null)
00706                 {
00707                     return attempt;
00708                 }
00709             }
00710
00711             return DefaultLibrary.ResolveFontFamily(standardFontFamily);
00712         }
00713     }
00714
00715 }
```

8.64 FormattedText.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
```

```
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021 using System.Linq;
00022 using static System.Net.Mime.MediaTypeNames;
00023
00024 namespace VectSharp
00025 {
00026 /// <summary>
00027 /// Represents the position of the text.
00028 /// </summary>
00029     public enum Script
00030     {
00031 /// <summary>
00032 /// The text is normal text.
00033 /// </summary>
00034         Normal,
00035
00036 /// <summary>
00037 /// The text is a superscript.
00038 /// </summary>
00039         Superscript,
00040
00041 /// <summary>
00042 /// The text is a subscript.
00043 /// </summary>
00044         Subscript
00045     }
00046
00047 /// <summary>
00048 /// Represents a run of text that should be drawn with the same style.
00049 /// </summary>
00050     public class FormattedText
00051     {
00052 /// <summary>
00053 /// Represents the text represented by this instance.
00054 /// </summary>
00055         public string Text { get; }
00056
00057 /// <summary>
00058 /// Represents the font that should be used to draw the text.
00059 /// </summary>
00060         public Font Font { get; }
00061
00062 /// <summary>
00063 /// Represents the position of the text.
00064 /// </summary>
00065         public Script Script { get; }
00066
00067 /// <summary>
00068 /// Represents the brush that should be used to draw the text.  If this is null, the default brush is
      used.
00069 /// </summary>
00070         public Brush Brush { get; }
00071
00072 /// <summary>
00073 /// Creates a new <see cref="FormattedText"/> instance with the specified <paramref name="text"/>,
      <paramref name="font"/>, <paramref name="script"/> position and <paramref name="brush"/>.
00074 /// </summary>
00075 /// <param name="text">The text that will be contained in the new <see cref="FormattedText"/>.</param>
00076 /// <param name="font">The font that will be used by the new <see cref="FormattedText"/>.</param>
00077 /// <param name="script">The script position of the new <see cref="FormattedText"/>.</param>
00078 /// <param name="brush">The brush that will be used by the new <see cref="FormattedText"/>.</param>
00079         public FormattedText(string text, Font font, Script script = Script.Normal, Brush brush =
      null)
00080         {
00081             this.Text = text ??  "";
00082             this.Font = font;
00083             this.Script = script;
00084             this.Brush = brush;
00085         }
00086
00087 /// <summary>
00088 /// Parse the formatting information contained in a text string into a collection of <see
      cref="FormattedText"/> objects.
00089 /// </summary>
00090 /// <param name="text">
00091 /// The string containing formatting information.  Format information is specified using HTML-like
      tags:
00092 /// <list type="bullet">
00093 /// <item><c>&lt;b&gt;&lt;/b&gt;</c> or <c>&lt;strong&gt;&lt;/strong&gt;</c> are used for bold
      text;</item>
00094 /// <item><c>&lt;i&gt;&lt;/i&gt;</c> or <c>&lt;em&gt;&lt;/em&gt;</c> are used for text in
      italics;</item>
```

```
00095 /// <item><c>&lt;u&gt;&lt;/u&gt;</c></item> are used for underlined text;
00096 /// <item><c>&lt;sup&gt;&lt;/sup&gt;</c> and <c>&lt;sub&gt;&lt;/sub&gt;</c> are used, respectively,
      for superscript and subscript text;</item>
00097 /// <item><c>&lt;#COLOUR&gt;&lt;/#&gt;</c> is used to specify the colour of the text, where
      <c>COLOUR</c> is a CSS colour string (e.g.  <c>&lt;#red&gt;</c>, <c>&lt;#0080FF&gt;</c>, or
      <c>&lt;#rgba(128, 80, 52, 0.5)&gt;</c>).</item>
00098 /// </list></param>
00099 /// <param name="normalFont">The font that will be used for text that is neither bold nor
      italic.</param>
00100 /// <param name="boldFont">The font that will be used for text that is bold.  Note that this does not
      necessarily have to be a bold font; this is just the font that is applied to text contained within
      <c>&lt;b&gt;&lt;/b&gt;</c> tags.</param>
00101 /// <param name="italicFont">The font that will be used for text that is in italics.  Note that this
      does not necessarily have to be an italic font; this is just the font that is applied to text
      contained within <c>&lt;i&gt;&lt;/i&gt;</c> tags.</param>
00102 /// <param name="boldItalicFont">The font that will be used for text that is both in bold and in
      italics.</param>
00103 /// <param name="defaultBrush">The default <see cref="Brush"/> that will be used for text runs that do
      not specify a colour.  If this is <see langword="null"/>, the default <see cref="Brush"/> will be the
      one specified in the painting call.</param>
00104 /// <returns>A lazy collection of <see cref="FormattedText"/> objects.  Note that every enumeration of
      this collection causes the text to be parsed again; if you need to enumerate this collection more than
      once, you should probably convert it e.g.  to a <see cref="List{T}"/>.</returns>
00105         public static IEnumerable<FormattedText> Format(string text, Font normalFont, Font boldFont,
      Font italicFont, Font boldItalicFont, Brush defaultBrush = null)
00106         {
00107             if (string.IsNullOrEmpty(text))
00108             {
00109                 yield break;
00110             }
00111 
00112             StringBuilder currentRun = new StringBuilder();
00113             int boldDepth = 0;
00114             int italicsDepth = 0;
00115             int underlineDepth = 0;
00116             int superscriptDepth = 0;
00117             int subscriptDepth = 0;
00118             Stack<Brush> brushes = new Stack<Brush>();
00119             brushes.Push(defaultBrush);
00120 
00121             for (int i = 0; i < text.Length; i++)
00122             {
00123                 if (text[i] != '<')
00124                 {
00125                     currentRun.Append(text[i]);
00126                 }
00127                 else
00128                 {
00129                     Tags tag = GetTag(text, i, out int tagEnd, out Brush tagBrush);
00130 
00131                     if (tag == Tags.None)
00132                     {
00133                         currentRun.Append(text[i]);
00134                     }
00135                     else
00136                     {
00137                         i = tagEnd;
00138 
00139                         string txt = currentRun.ToString();
00140 
00141                         switch (tag)
00142                         {
00143                             case Tags.BoldOpen:
00144                                 if (!string.IsNullOrEmpty(txt))
00145                                 {
00146                                     yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00147                                 }
00148                                 currentRun.Clear();
00149                                 boldDepth++;
00150                                 break;
00151                             case Tags.BoldClose:
00152                                 if (boldDepth > 0)
00153                                 {
00154                                     if (!string.IsNullOrEmpty(txt))
00155                                     {
00156                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00157                                     }
00158                                     currentRun.Clear();
00159                                     boldDepth--;
00160                                 }
00161                                 break;
```

```
00162                           case Tags.UnderlineOpen:
00163                               if (!string.IsNullOrEmpty(txt))
00164                               {
00165                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00166                               }
00167                               currentRun.Clear();
00168                               underlineDepth++;
00169                               break;
00170                           case Tags.UnderlineClose:
00171                               if (underlineDepth > 0)
00172                               {
00173                                   if (!string.IsNullOrEmpty(txt))
00174                                   {
00175                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00176                                   }
00177                                   currentRun.Clear();
00178                                   underlineDepth--;
00179                               }
00180                               break;
00181
00182                           case Tags.ItalicsOpen:
00183                               if (!string.IsNullOrEmpty(txt))
00184                               {
00185                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00186                               }
00187                               currentRun.Clear();
00188                               italicsDepth++;
00189                               break;
00190                           case Tags.ItalicsClose:
00191                               if (italicsDepth > 0)
00192                               {
00193                                   if (!string.IsNullOrEmpty(txt))
00194                                   {
00195                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00196                                   }
00197                                   currentRun.Clear();
00198                                   italicsDepth--;
00199                               }
00200                               break;
00201
00202                           case Tags.SupOpen:
00203                               if (!string.IsNullOrEmpty(txt))
00204                               {
00205                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00206                               }
00207                               currentRun.Clear();
00208                               superscriptDepth++;
00209                               break;
00210                           case Tags.SupClose:
00211                               if (superscriptDepth > 0)
00212                               {
00213                                   if (!string.IsNullOrEmpty(txt))
00214                                   {
00215                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00216                                   }
00217                                   currentRun.Clear();
00218                                   superscriptDepth--;
00219                               }
00220                               break;
00221
00222                           case Tags.SubOpen:
00223                               if (!string.IsNullOrEmpty(txt))
00224                               {
00225                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00226                               }
00227                               currentRun.Clear();
```

```
00228                                 subscriptDepth++;
00229                                 break;
00230                             case Tags.SubClose:
00231                                 if (subscriptDepth > 0)
00232                                 {
00233                                     if (!string.IsNullOrEmpty(txt))
00234                                     {
00235                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00236                                     }
00237                                     currentRun.Clear();
00238                                     subscriptDepth--;
00239                                 }
00240                                 break;
00241                             case Tags.ColourOpen:
00242                                 if (!string.IsNullOrEmpty(txt))
00243                                 {
00244                                     yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00245                                 }
00246                                 currentRun.Clear();
00247                                 brushes.Push(tagBrush);
00248                                 break;
00249                             case Tags.ColourClose:
00250                                 if (brushes.Count > 1)
00251                                 {
00252                                     if (!string.IsNullOrEmpty(txt))
00253                                     {
00254                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00255                                     }
00256                                     currentRun.Clear();
00257                                     brushes.Pop();
00258                                 }
00259                                 break;
00260                         }
00261                     }
00262                 }
00263             }
00264
00265             if (currentRun.Length > 0)
00266             {
00267                 yield return new FormattedText(currentRun.ToString(), GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek());
00268             }
00269         }
00270
00271 /// <summary>
00272 /// Parse the formatting information contained in a text string into a collection of <see
      cref="FormattedText"/> objects, using fonts from a standard font family.
00273 /// </summary>
00274 /// <param name="text">The string containing formatting information.  Format information is specified
      using HTML-like tags:
00275 /// <list type="bullet">
00276 /// <item><c>&lt;b&gt;&lt;/b&gt;</c> or <c>&lt;strong&gt;&lt;/strong&gt;</c> are used for bold
      text;</item>
00277 /// <item><c>&lt;i&gt;&lt;/i&gt;</c> or <c>&lt;em&gt;&lt;/em&gt;</c> are used for text in
      italics;</item>
00278 /// <item><c>&lt;u&gt;&lt;/u&gt;</c></item> are used for underlined text;
00279 /// <item><c>&lt;sup&gt;&lt;/sup&gt;</c> and <c>&lt;sub&gt;&lt;/sub&gt;</c> are used, respectively,
      for superscript and subscript text;</item>
00280 /// <item><c>&lt;#COLOUR&gt;&lt;/#&gt;</c> is used to specify the colour of the text, where
      <c>COLOUR</c> is a CSS colour string (e.g.  <c>&lt;#red&gt;</c>, <c>&lt;#0080FF&gt;</c>, or
      <c>&lt;#rgba(128, 80, 52, 0.5)&gt;</c>).</item>
00281 /// </list></param>
00282 /// <param name="fontFamily">The font family from which the fonts will be created.  If this is a
      regular font family, the bold, italic and bold-italic versions of the font will be used for the
      formatted text.  Otherwise, the relevant font styles will be toggled (e.g.  if the supplied font
      family is bold, then regular text in the formatted string will be displayed as bold, while bold text
      in the formatted string will be displayed as regular text).</param>
00283 /// <param name="fontSize">The size of the fonts to use.</param>
00284 /// <param name="defaultUnderline">Determines whether text should be underlined by default.  This is
      toggled by <c>&lt;u&gt;&lt;/u&gt;</c> tags.</param>
00285 /// <param name="defaultBrush">The default <see cref="Brush"/> that will be used for text runs that do
      not specify a colour.  If this is <see langword="null"/>, the default <see cref="Brush"/> will be the
      one specified in the painting call.</param>
00286 /// <returns>A lazy collection of <see cref="FormattedText"/> objects.  Note that every enumeration of
      this collection causes the text to be parsed again; if you need to enumerate this collection more than
      once, you should probably convert it e.g.  to a <see cref="List{T}"/>.</returns>
```

```
00287         public static IEnumerable<FormattedText> Format(string text, FontFamily.StandardFontFamilies
      fontFamily, double fontSize, bool defaultUnderline = false, Brush defaultBrush = null)
00288         {
00289             Font normalFont = new Font(FontFamily.ResolveFontFamily(fontFamily), fontSize,
      defaultUnderline);
00290
00291             Font boldFont = normalFont;
00292             Font italicFont = normalFont;
00293             Font boldItalicFont = normalFont;
00294
00295             switch (fontFamily)
00296             {
00297                 case FontFamily.StandardFontFamilies.Courier:
00298                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
      defaultUnderline);
00299                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
      defaultUnderline);
00300                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
      defaultUnderline);
00301                     break;
00302                 case FontFamily.StandardFontFamilies.CourierBold:
00303                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
      defaultUnderline);
00304                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
      defaultUnderline);
00305                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
      defaultUnderline);
00306                     break;
00307                 case FontFamily.StandardFontFamilies.CourierOblique:
00308                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
      defaultUnderline);
00309                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
      defaultUnderline);
00310                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
      defaultUnderline);
00311                     break;
00312                 case FontFamily.StandardFontFamilies.CourierBoldOblique:
00313                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
      defaultUnderline);
00314                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
      defaultUnderline);
00315                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
      defaultUnderline);
00316                     break;
00317
00318                 case FontFamily.StandardFontFamilies.Helvetica:
00319                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
      defaultUnderline);
00320                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
      defaultUnderline);
00321                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00322                     break;
00323                 case FontFamily.StandardFontFamilies.HelveticaBold:
00324                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
      defaultUnderline);
00325                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00326                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
      defaultUnderline);
00327                     break;
00328                 case FontFamily.StandardFontFamilies.HelveticaOblique:
00329                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00330                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
      defaultUnderline);
00331                     boldItalicFont = new
```

```
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
defaultUnderline);
00332                     break;
00333                 case FontFamily.StandardFontFamilies.HelveticaBoldOblique:
00334                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
defaultUnderline);
00335                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
defaultUnderline);
00336                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
defaultUnderline);
00337                     break;
00338
00339                 case FontFamily.StandardFontFamilies.TimesRoman:
00340                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
defaultUnderline);
00341                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
defaultUnderline);
00342                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
defaultUnderline);
00343                     break;
00344                 case FontFamily.StandardFontFamilies.TimesBold:
00345                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
defaultUnderline);
00346                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
defaultUnderline);
00347                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
defaultUnderline);
00348                     break;
00349                 case FontFamily.StandardFontFamilies.TimesItalic:
00350                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
defaultUnderline);
00351                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
defaultUnderline);
00352                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
defaultUnderline);
00353                     break;
00354                 case FontFamily.StandardFontFamilies.TimesBoldItalic:
00355                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
defaultUnderline);
00356                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
defaultUnderline);
00357                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
defaultUnderline);
00358                     break;
00359             }
00360
00361             return Format(text, normalFont, boldFont, italicFont, boldItalicFont, defaultBrush);
00362         }
00363
00364         private static Font GetFont(bool bold, bool italic, bool underlined, Font normalFont, Font
boldFont, Font italicFont, Font boldItalicFont)
00365         {
00366             if (!bold && !italic)
00367             {
00368                 if (underlined)
00369                 {
00370                     return new Font(normalFont.FontFamily, normalFont.FontSize, normalFont.Underline
== null);
00371                 }
00372                 else
00373                 {
00374                     return normalFont;
00375                 }
00376             }
00377             else if (bold && !italic)
00378             {
00379                 if (underlined)
00380                 {
00381                     return new Font(boldFont.FontFamily, boldFont.FontSize, boldFont.Underline ==
null);
00382                 }
00383                 else
```

```
00384                 {
00385                     return boldFont;
00386                 }
00387             }
00388             else if (!bold && italic)
00389             {
00390                 if (underlined)
00391                 {
00392                     return new Font(italicFont.FontFamily, italicFont.FontSize, italicFont.Underline
      == null);
00393                 }
00394                 else
00395                 {
00396                     return italicFont;
00397                 }
00398             }
00399             else
00400             {
00401                 if (underlined)
00402                 {
00403                     return new Font(boldItalicFont.FontFamily, boldItalicFont.FontSize,
      boldItalicFont.Underline == null);
00404                 }
00405                 else
00406                 {
00407                     return boldItalicFont;
00408                 }
00409             }
00410         }
00411
00412         private enum Tags
00413         {
00414             BoldOpen,
00415             BoldClose,
00416             ItalicsOpen,
00417             ItalicsClose,
00418             UnderlineOpen,
00419             UnderlineClose,
00420             SupOpen,
00421             SupClose,
00422             SubOpen,
00423             SubClose,
00424             ColourOpen,
00425             ColourClose,
00426             None
00427         }
00428
00429         private static Tags GetTag(string text, int start, out int tagEnd, out Brush tagBrush)
00430         {
00431             StringBuilder tag = new StringBuilder();
00432             bool closed = false;
00433
00434             int i = start;
00435
00436             for (; i < text.Length; i++)
00437             {
00438                 tag.Append(text[i]);
00439
00440                 if (text[i] == '>')
00441                 {
00442                     closed = true;
00443                     break;
00444                 }
00445             }
00446
00447             tagEnd = i;
00448             tagBrush = null;
00449
00450             if (!closed)
00451             {
00452                 return Tags.None;
00453             }
00454             else
00455             {
00456                 string tagString = tag.Replace(" ", "").ToString().ToLowerInvariant();
00457
00458                 if (tagString == "<b>" || tagString == "<strong>")
00459                 {
00460                     return Tags.BoldOpen;
00461                 }
00462                 else if (tagString == "</b>" || tagString == "</strong>")
00463                 {
00464                     return Tags.BoldClose;
00465                 }
00466                 else if (tagString == "<i>" || tagString == "<em>")
00467                 {
00468                     return Tags.ItalicsOpen;
```

```
00469                 }
00470                 else if (tagString == "</i>" || tagString == "</em>")
00471                 {
00472                     return Tags.ItalicsClose;
00473                 }
00474                 else if (tagString == "<u>")
00475                 {
00476                     return Tags.UnderlineOpen;
00477                 }
00478                 else if (tagString == "</u>")
00479                 {
00480                     return Tags.UnderlineClose;
00481                 }
00482                 else if (tagString == "<sup>")
00483                 {
00484                     return Tags.SupOpen;
00485                 }
00486                 else if (tagString == "</sup>")
00487                 {
00488                     return Tags.SupClose;
00489                 }
00490                 else if (tagString == "<sub>")
00491                 {
00492                     return Tags.SubOpen;
00493                 }
00494                 else if (tagString == "</sub>")
00495                 {
00496                     return Tags.SubClose;
00497                 }
00498                 else if (tagString.StartsWith("<#"))
00499                 {
00500                     string colour = tagString.Substring(1, tagString.Length - 2);
00501 
00502                     Colour?  col = null;
00503 
00504                     try
00505                     {
00506                         col = Colour.FromCSSString(colour);
00507                     }
00508                     catch { }
00509 
00510                     if (col == null)
00511                     {
00512                         colour = tagString.Substring(2, tagString.Length - 3);
00513 
00514                         col = Colour.FromCSSString(colour);
00515                     }
00516 
00517                     if (col != null)
00518                     {
00519                         tagBrush = col.Value;
00520                         return Tags.ColourOpen;
00521                     }
00522                     else
00523                     {
00524                         return Tags.None;
00525                     }
00526                 }
00527                 else if (tagString == "</#>")
00528                 {
00529                     return Tags.ColourClose;
00530                 }
00531                 else
00532                 {
00533                     return Tags.None;
00534                 }
00535             }
00536         }
00537     }
00538 
00539 /// <summary>
00540 /// Contains extension methods for collections of <see cref="FormattedText"/> objects.
00541 /// </summary>
00542     public static class FormattedTextExtensions
00543     {
00544         internal static Font.DetailedFontMetrics Measure(this IEnumerable<FormattedText> text,
      List<FormattedText> items, List<Font.DetailedFontMetrics> allMetrics)
00545         {
00546             double width = 0;
00547             double advanceWidth = 0;
00548 
00549             double lsb = 0;
00550             double rsb = 0;
00551 
00552             double top = 0;
00553             double bottom = 0;
00554 
```

```
00555             bool isFirst = true;
00556
00557             foreach (FormattedText txt in text)
00558             {
00559                 if (text != null && !string.IsNullOrEmpty(txt.Text))
00560                 {
00561                     items?.Add(txt);
00562
00563                     if (txt.Script == Script.Normal)
00564                     {
00565                         Font.DetailedFontMetrics metrics = txt.Font.MeasureTextAdvanced(txt.Text);
00566                         allMetrics?.Add(metrics);
00567
00568                         top = Math.Max(top, metrics.Top);
00569                         bottom = Math.Min(bottom, metrics.Bottom);
00570                         rsb = metrics.RightSideBearing;
00571
00572                         advanceWidth += metrics.AdvanceWidth;
00573
00574                         if (!isFirst)
00575                         {
00576                             width += metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing;
00577                         }
00578                         else
00579                         {
00580                             width += metrics.Width + metrics.RightSideBearing;
00581                             lsb = metrics.LeftSideBearing;
00582                         }
00583                     }
00584                     else
00585                     {
00586                         Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7);
00587
00588                         Font.DetailedFontMetrics metrics = newFont.MeasureTextAdvanced(txt.Text);
00589                         allMetrics?.Add(metrics);
00590
00591                         advanceWidth += metrics.AdvanceWidth;
00592
00593                         if (txt.Script == Script.Subscript)
00594                         {
00595                             top = Math.Max(top, metrics.Top - txt.Font.FontSize * 0.14);
00596                             bottom = Math.Min(bottom, metrics.Bottom - txt.Font.FontSize * 0.14);
00597                         }
00598                         else if (txt.Script == Script.Superscript)
00599                         {
00600                             top = Math.Max(top, metrics.Top + txt.Font.FontSize * 0.33);
00601                             bottom = Math.Min(bottom, metrics.Bottom + txt.Font.FontSize * 0.33);
00602                         }
00603
00604                         rsb = metrics.RightSideBearing;
00605
00606                         if (!isFirst)
00607                         {
00608                             width += metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing;
00609                         }
00610                         else
00611                         {
00612                             width += metrics.Width + metrics.RightSideBearing;
00613                             lsb = metrics.LeftSideBearing;
00614                         }
00615                     }
00616
00617                     if (isFirst)
00618                     {
00619                         isFirst = false;
00620                     }
00621                 }
00622             }
00623
00624             width -= rsb;
00625
00626             return new Font.DetailedFontMetrics(width, top - bottom, lsb, rsb, top, bottom,
      advanceWidth);
00627         }
00628
00629 /// <summary>
00630 /// Measures a collection of <see cref="FormattedText"/> objects.
00631 /// </summary>
00632 /// <param name="text">The collection of <see cref="FormattedText"/> objects to be measured.</param>
00633 /// <returns>A <see cref="Font.DetailedFontMetrics"/> containing detailed measurements for the text
      obtained by composing the elements in the <see cref="FormattedText"/> collection.</returns>
00634         public static Font.DetailedFontMetrics Measure(this IEnumerable<FormattedText> text)
00635         {
00636             return text.Measure(null, null);
00637         }
```

```
00638
00639 /// <summary>
00640 /// Extracts the text from a collection of <see cref="FormattedText"/> objects.
00641 /// </summary>
00642 /// <param name="text">The collection of <see cref="FormattedText"/> objects whose text should be
      extracted.</param>
00643 /// <returns>A text rappresentation of the collection of <see cref="FormattedText"/>
      objects.</returns>
00644         public static string GetText(this IEnumerable<FormattedText> text)
00645         {
00646             StringBuilder sb = new StringBuilder();
00647
00648             foreach (FormattedText txt in text)
00649             {
00650                 sb.Append(txt.ToString());
00651             }
00652
00653             return sb.ToString();
00654         }
00655     }
00656
00657 }
```

8.65 Gradients.cs

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using VectSharp;
00006 using static System.Net.WebRequestMethods;
00007
00008 namespace VectSharp
00009 {
00010 /// <summary>
00011 /// Standard gradients.
00012 /// </summary>
00013     public static class Gradients
00014     {
00015 /// <summary>
00016 /// Gradient from transparent black (0) to opaque black (1).
00017 /// </summary>
00018         public static readonly GradientStops TransparentToBlack = new GradientStops(new
      GradientStop(Colour.FromRgba(0, 0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1));
00019
00020 /// <summary>
00021 /// Gradient from white (0) to black (1).
00022 /// </summary>
00023         public static readonly GradientStops WhiteToBlack = new GradientStops(new
      GradientStop(Colour.FromRgb(255, 255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1));
00024
00025 /// <summary>
00026 /// Gradient from red (0) to green (1).
00027 /// </summary>
00028         public static readonly GradientStops RedToGreen = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1));
00029
00030 /// <summary>
00031 /// Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).
00032 /// </summary>
00033         public static readonly GradientStops Rainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6),
      new GradientStop(Colour.FromRgb(255, 242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76),
      0.5), new GradientStop(Colour.FromRgb(0, 162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72,
      204), 5.0 / 6), new GradientStop(Colour.FromRgb(163, 73, 164), 1));
00034
00035 /// <summary>
00036 /// Gradient from red (0) to yellow (0.5) to green (1).
00037 /// </summary>
00038         public static readonly GradientStops RedYellowGreen = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new
      GradientStop(Colour.FromRgb(34, 177, 76), 1));
00039
00040 /// <summary>
00041 /// Rainbow gradient with Okabe-Ito colour-blind safe colours (<seealso
      href="https://jfly.uni-koeln.de/color/"/>).
00042 /// </summary>
00043         public static readonly GradientStops OkabeItoRainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6),
      new GradientStop(Colour.FromRgb(230, 159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66),
      0.5), new GradientStop(Colour.FromRgb(0, 158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114,
      178), 5.0 / 6), new GradientStop(Colour.FromRgb(86, 180, 233), 1));
00044
```

```
00045 /// <summary>
00046 /// Rainbow gradient with Okabe-Ito colour-blind safe colours (<seealso
      href="https://jfly.uni-koeln.de/color/"/>) in discrete steps.
00047 /// </summary>
00048         public static readonly GradientStops OkabeItoRainbowDiscrete = new GradientStops(new
      GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121, 167), 0.2),
      new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230, 159, 0),
      0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0, 158,
      115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0,
      114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new
      GradientStop(Colour.FromRgb(86, 180, 233), 1));
00049
00050 /// <summary>
00051 /// Rainbow gradient with Paul Tol's Muted palette (<seealso
      href="https://personal.sron.nl/~pault/"/>).
00052 /// </summary>
00053         public static readonly GradientStops MutedRainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(221, 204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125),
      new GradientStop(Colour.FromRgb(17, 119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153),
      0.375), new GradientStop(Colour.FromRgb(136, 204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34,
      136), 0.625), new GradientStop(Colour.FromRgb(170, 68, 153), 0.75), new
      GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204, 102, 119), 1));
00054
00055 /// <summary>
00056 /// Rainbow gradient with Paul Tol's Muted palette (<seealso
      href="https://personal.sron.nl/~pault/"/>) in discrete steps.
00057 /// </summary>
00058         public static readonly GradientStops MutedRainbowDiscrete = new GradientStops(new
      GradientStop(Colour.FromRgb(187, 187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187),
      0.1), new GradientStop(Colour.FromRgb(221, 204, 119), 0.101), new GradientStop(Colour.FromRgb(221,
      204, 119), 0.2), new GradientStop(Colour.FromRgb(153, 153, 51), 0.201), new
      GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17, 119, 51), 0.301),
      new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68, 170, 153),
      0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136, 204,
      238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new
      GradientStop(Colour.FromRgb(51, 34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7),
      new GradientStop(Colour.FromRgb(170, 68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153),
      0.8), new GradientStop(Colour.FromRgb(136, 34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34,
      85), 0.9), new GradientStop(Colour.FromRgb(204, 102, 119), 0.901), new
      GradientStop(Colour.FromRgb(204, 102, 119), 1));
00059
00060         private static double[,] MagmaRGB = new double[256, 3] { { 0.00146159096, 0.000466127766,
      0.01386552 }, { 0.00225764007, 0.00129495431, 0.0183311461 }, { 0.00327943222, 0.00230452991,
      0.0237083291 }, { 0.00451230222, 0.00349037666, 0.0299647059 }, { 0.00594976987, 0.00484285,
      0.0371296695 }, { 0.0075879855, 0.00635613622, 0.0449730774 }, { 0.0094260439, 0.00802185006,
      0.0528443561 }, { 0.0114654337, 0.00982831486, 0.060749638 }, { 0.0137075706, 0.0117705913,
      0.0686665843 }, { 0.0161557566, 0.0138404966, 0.076602666 }, { 0.018815367, 0.0160262753, 0.0845844897
      }, { 0.021691934, 0.0183201254, 0.092610105 }, { 0.0247917814, 0.0207147875, 0.100675555 }, {
      0.0281228154, 0.0232009284, 0.108786954 }, { 0.0316955304, 0.0257651161, 0.116964722 }, {
      0.0355204468, 0.028397457, 0.125209396 }, { 0.0396084872, 0.0310895652, 0.133515085 }, { 0.043829535,
      0.0338299885, 0.141886249 }, { 0.0480616391, 0.0366066101, 0.150326989 }, { 0.0523204388, 0.039406602,
      0.158841025 }, { 0.0566148978, 0.0421598925, 0.167445592 }, { 0.060949393, 0.0447944924, 0.176128834
      }, { 0.0653301801, 0.0473177796, 0.184891506 }, { 0.0697637296, 0.0497264666, 0.193735088 }, {
      0.0742565152, 0.0520167766, 0.202660374 }, { 0.0788150034, 0.0541844801, 0.211667355 }, {
      0.0834456313, 0.0562249365, 0.220755099 }, { 0.088154773, 0.0581331465, 0.229921611 }, { 0.0929486914,
      0.0599038167, 0.239163669 }, { 0.097833477, 0.0615314414, 0.248476662 }, { 0.102814972, 0.0630104053,
      0.2578544 }, { 0.107898679, 0.0643351102, 0.267288933 }, { 0.113094451, 0.0654920358, 0.276783978 }, {
      0.118405035, 0.0664791593, 0.286320656 }, { 0.123832651, 0.0672946449, 0.295879431 }, { 0.129380192,
      0.0679349264, 0.305442931 }, { 0.135053322, 0.0683912798, 0.31499989 }, { 0.140857952, 0.068654071,
      0.32453764 }, { 0.146785234, 0.0687382323, 0.334011109 }, { 0.152839217, 0.0686368599, 0.34340445 }, {
      0.159017511, 0.0683540225, 0.352688028 }, { 0.165308131, 0.0679108689, 0.361816426 }, { 0.171713033,
      0.067305326, 0.370770827 }, { 0.17821173, 0.0665758073, 0.379497161 }, { 0.184800877, 0.0657324381,
      0.387972507 }, { 0.191459745, 0.0648183312, 0.396151969 }, { 0.198176877, 0.0638624166, 0.404008953 },
      { 0.204934882, 0.0629066192, 0.411514273 }, { 0.211718061, 0.0619917876, 0.418646741 }, { 0.21851159,
      0.0611584918, 0.425391816 }, { 0.225302032, 0.0604451843, 0.431741767 }, { 0.232076515, 0.0598886855,
      0.437694665 }, { 0.238825991, 0.0595170384, 0.443255999 }, { 0.245543175, 0.0593524384, 0.448435938 },
      { 0.252220252, 0.0594147119, 0.453247729 }, { 0.258857304, 0.0597055998, 0.457709924 }, { 0.265446744,
      0.0602368754, 0.461840297 }, { 0.271994089, 0.0609935552, 0.465660375 }, { 0.2784933, 0.0619778136,
      0.469190328 }, { 0.284951097, 0.0631676261, 0.472450879 }, { 0.291365817, 0.0645534486, 0.475462193 },
      { 0.297740413, 0.0661170432, 0.478243482 }, { 0.304080941, 0.0678353452, 0.480811572 }, { 0.310382027,
      0.0697024767, 0.48318634 }, { 0.316654235, 0.0716895272, 0.485380429 }, { 0.322899126, 0.0737819504,
      0.487408399 }, { 0.329114038, 0.0759715081, 0.489286796 }, { 0.335307503, 0.0782361045, 0.491024144 },
      { 0.341481725, 0.0805635079, 0.492631321 }, { 0.347635742, 0.0829463512, 0.494120923 }, { 0.353773161,
      0.0853726329, 0.495501096 }, { 0.359897941, 0.0878311772, 0.496778331 }, { 0.366011928, 0.0903143031,
      0.497959963 }, { 0.372116205, 0.0928159917, 0.499053326 }, { 0.378210547, 0.0953322947, 0.500066568 },
      { 0.384299445, 0.0978549106, 0.501001964 }, { 0.390384361, 0.100379466, 0.501864236 }, { 0.39646667,
      0.102902194, 0.50265759 }, { 0.402547663, 0.105419865, 0.503385761 }, { 0.408628505, 0.107929771,
      0.504052118 }, { 0.414708664, 0.110431177, 0.504661843 }, { 0.420791157, 0.11292021, 0.505214935 }, {
      0.426876965, 0.115395258, 0.505713602 }, { 0.432967001, 0.117854987, 0.506159754 }, { 0.439062114,
      0.120298314, 0.506555026 }, { 0.445163096, 0.122724371, 0.506900806 }, { 0.451270678, 0.125132484,
      0.507198258 }, { 0.457385535, 0.127522145, 0.507448336 }, { 0.463508291, 0.129892998, 0.507651812 }, {
      0.469639514, 0.132244819, 0.507809282 }, { 0.475779723, 0.1345775, 0.507921193 }, { 0.481928997,
      0.13689139, 0.507988509 }, { 0.488088169, 0.139186217, 0.508010737 }, { 0.494257673, 0.141462106,
      0.507987836 }, { 0.500437834, 0.143719323, 0.507919772 }, { 0.506628929, 0.145958202, 0.50780642 }, {
      0.512831195, 0.148179144, 0.50764757 }, { 0.519044825, 0.150382611, 0.507442938 }, { 0.525269968,
      0.152569121, 0.507192172 }, { 0.531506735, 0.154739247, 0.50689486 }, { 0.537755194, 0.156893613,
      0.506550538 }, { 0.544015371, 0.159032895, 0.506158696 }, { 0.550287252, 0.161157816, 0.505718782 }, {
```

```
        0.556570783, 0.163269149, 0.50523021 }, { 0.562865867, 0.165367714, 0.504692365 }, { 0.569172368,
        0.167454379, 0.504104606 }, { 0.575490107, 0.169530062, 0.503466273 }, { 0.581818864, 0.171595728,
        0.50277669 }, { 0.588158375, 0.173652392, 0.502035167 }, { 0.594508337, 0.175701122, 0.501241011 }, {
        0.600868399, 0.177743036, 0.500393522 }, { 0.607238169, 0.179779309, 0.499491999 }, { 0.613617209,
        0.18181117, 0.498535746 }, { 0.620005032, 0.183839907, 0.497524075 }, { 0.626401108, 0.185866869,
        0.496456304 }, { 0.632804854, 0.187893468, 0.495331769 }, { 0.639215638, 0.189921182, 0.494149821 }, {
        0.645632778, 0.191951556, 0.492909832 }, { 0.652055535, 0.19398621, 0.491611196 }, { 0.658483116,
        0.196026835, 0.490253338 }, { 0.664914668, 0.198075202, 0.488835712 }, { 0.671349279, 0.200133166,
        0.487357807 }, { 0.677785975, 0.202202663, 0.485819154 }, { 0.684223712, 0.204285721, 0.484219325 }, {
        0.69066138, 0.206384461, 0.482557941 }, { 0.697097796, 0.2085011, 0.480834678 }, { 0.7035317,
        0.210637956, 0.47904927 }, { 0.709961888, 0.212797337, 0.477201121 }, { 0.716387038, 0.214981693,
        0.47528978 }, { 0.722805451, 0.217193831, 0.473315708 }, { 0.729215521, 0.219436516, 0.471278924 }, {
        0.735615545, 0.221712634, 0.469179541 }, { 0.742003713, 0.224025196, 0.467017774 }, { 0.748378107,
        0.226377345, 0.464793954 }, { 0.754736692, 0.228772352, 0.462508534 }, { 0.761077312, 0.231213625,
        0.460162106 }, { 0.767397681, 0.233704708, 0.457755411 }, { 0.77369538, 0.236249283, 0.455289354 }, {
        0.779967847, 0.23885117, 0.452765022 }, { 0.786212372, 0.241514325, 0.450183695 }, { 0.792426972,
        0.24424225, 0.447543155 }, { 0.79860776, 0.247039798, 0.444848441 }, { 0.804751511, 0.24991135,
        0.442101615 }, { 0.810854841, 0.252861399, 0.439304963 }, { 0.816914186, 0.25589455, 0.436461074 }, {
        0.822925797, 0.259015505, 0.433572874 }, { 0.82888574, 0.262229049, 0.430643647 }, { 0.834790818,
        0.265539703, 0.427671352 }, { 0.84063568, 0.268952874, 0.42466562 }, { 0.846415804, 0.272473491,
        0.421631064 }, { 0.85212649, 0.276106469, 0.418572767 }, { 0.85776287, 0.279856666, 0.415496319 }, {
        0.863320397, 0.283729003, 0.412402889 }, { 0.868793368, 0.287728205, 0.409303002 }, { 0.874176342,
        0.291858679, 0.406205397 }, { 0.879463944, 0.296124596, 0.403118034 }, { 0.884650824, 0.30053009,
        0.40004706 }, { 0.889731418, 0.305078817, 0.397001559 }, { 0.894700194, 0.309773445, 0.393994634 }, {
        0.899551884, 0.314616425, 0.391036674 }, { 0.904281297, 0.319609981, 0.388136889 }, { 0.908883524,
        0.324755126, 0.385308008 }, { 0.913354091, 0.330051947, 0.382563414 }, { 0.917688852, 0.335500068,
        0.379915138 }, { 0.921884187, 0.341098112, 0.377375977 }, { 0.925937102, 0.346843685, 0.374959077 }, {
        0.92984509, 0.352733817, 0.372676513 }, { 0.933606454, 0.358764377, 0.370540883 }, { 0.937220874,
        0.364929312, 0.368566525 }, { 0.940687443, 0.371224168, 0.366761699 }, { 0.944006448, 0.377642889,
        0.365136328 }, { 0.947179528, 0.384177874, 0.36370113 }, { 0.95021015, 0.390819546, 0.362467694 }, {
        0.953099077, 0.397562894, 0.361438431 }, { 0.955849237, 0.404400213, 0.360619076 }, { 0.958464079,
        0.411323666, 0.360014232 }, { 0.960949221, 0.418323245, 0.359629789 }, { 0.963310281, 0.425389724,
        0.35946902 }, { 0.965549351, 0.432518707, 0.359529151 }, { 0.967671128, 0.439702976, 0.359810172 }, {
        0.969680441, 0.446935635, 0.36031112 }, { 0.971582181, 0.45421017, 0.361030156 }, { 0.973381238,
        0.461520484, 0.361964652 }, { 0.975082439, 0.468860936, 0.363111292 }, { 0.976690494, 0.47622635,
        0.364466162 }, { 0.978209957, 0.483612031, 0.366024854 }, { 0.979645181, 0.491013764, 0.367782559 }, {
        0.981000291, 0.4984278, 0.369734157 }, { 0.982279159, 0.505850848, 0.371874301 }, { 0.983485387,
        0.513280054, 0.374197501 }, { 0.984622298, 0.520712972, 0.376698186 }, { 0.985692925, 0.528147545,
        0.379370774 }, { 0.986700017, 0.53558207, 0.382209724 }, { 0.987646038, 0.543015173, 0.385209578 }, {
        0.988533173, 0.550445778, 0.388365009 }, { 0.989363341, 0.557873075, 0.391670846 }, { 0.990138201,
        0.565296495, 0.395122099 }, { 0.990871208, 0.572706259, 0.398713971 }, { 0.991558165, 0.580106828,
        0.402441058 }, { 0.992195728, 0.587501706, 0.406298792 }, { 0.992784669, 0.594891088, 0.410282976 }, {
        0.993325561, 0.602275297, 0.414389658 }, { 0.993834412, 0.60964354, 0.418613221 }, { 0.994308514,
        0.616998953, 0.422949672 }, { 0.994737698, 0.624349657, 0.427396771 }, { 0.995121854, 0.631696376,
        0.431951492 }, { 0.995480469, 0.639026596, 0.436607159 }, { 0.995809924, 0.646343897, 0.441360951 }, {
        0.996095703, 0.653658756, 0.446213021 }, { 0.996341406, 0.660969379, 0.451160201 }, { 0.996579803,
        0.668255621, 0.456191814 }, { 0.996774784, 0.675541484, 0.461314158 }, { 0.996925427, 0.682827953,
        0.466525689 }, { 0.997077185, 0.690087897, 0.471811461 }, { 0.997186253, 0.697348991, 0.477181727 }, {
        0.997253982, 0.704610791, 0.482634651 }, { 0.99732518, 0.711847714, 0.488154375 }, { 0.997350983,
        0.719089119, 0.493754665 }, { 0.997350583, 0.726324415, 0.499427972 }, { 0.997341259, 0.733544671,
        0.505166839 }, { 0.997284689, 0.740771893, 0.510983331 }, { 0.997228367, 0.747980563, 0.516859378 }, {
        0.99713848, 0.755189852, 0.522805996 }, { 0.997019342, 0.762397883, 0.528820775 }, { 0.996898254,
        0.769590975, 0.534892341 }, { 0.996726862, 0.77679486, 0.541038571 }, { 0.996570645, 0.783976508,
        0.547232992 }, { 0.996369065, 0.791167346, 0.553498939 }, { 0.996162309, 0.798347709, 0.559819643 }, {
        0.995932448, 0.805527126, 0.566201824 }, { 0.995680107, 0.812705773, 0.572644795 }, { 0.995423973,
        0.819875302, 0.57914013 }, { 0.995131288, 0.827051773, 0.585701463 }, { 0.994851089, 0.834212826,
        0.592307093 }, { 0.994523666, 0.841386618, 0.598982818 }, { 0.9942219, 0.848540474, 0.605695903 }, {
        0.993865767, 0.855711038, 0.612481798 }, { 0.993545285, 0.862858846, 0.6192993 }, { 0.993169558,
        0.870024467, 0.626189463 }, { 0.992830963, 0.877168404, 0.633109148 }, { 0.992439881, 0.884329694,
        0.640099465 }, { 0.992089454, 0.891469549, 0.647116021 }, { 0.991687744, 0.89862705, 0.654201544 }, {
        0.991331929, 0.905762748, 0.661308839 }, { 0.990929685, 0.91291501, 0.668481201 }, { 0.990569914,
        0.920048699, 0.675674592 }, { 0.990174637, 0.927195612, 0.682925602 }, { 0.989814839, 0.93432854,
        0.690198194 }, { 0.989433736, 0.941470354, 0.697518628 }, { 0.989077438, 0.948604077, 0.704862519 }, {
        0.988717064, 0.95574152, 0.712242232 }, { 0.988367028, 0.962878026, 0.719648627 }, { 0.988032885,
        0.970012413, 0.727076773 }, { 0.987690702, 0.977154231, 0.734536205 }, { 0.987386827, 0.984287561,
        0.742001547 }, { 0.987052509, 0.991437853, 0.749504188 } };
00061         private static double[,] InfernoRGB = new double[256, 3] { { 0.00146159096, 0.000466127766,
        0.01386552 }, { 0.00226726368, 0.00126992553, 0.018570352 }, { 0.00329899092, 0.00224934863,
        0.0242390508 }, { 0.00454690615, 0.00339180156, 0.0309092475 }, { 0.00600552565, 0.00469194561,
        0.038557898 }, { 0.00767578856, 0.00613611626, 0.0468360336 }, { 0.00956051094, 0.00771344131,
        0.0551430756 }, { 0.0116634769, 0.00941675403, 0.063459808 }, { 0.0139950388, 0.0112247138,
        0.071861689 }, { 0.0165605595, 0.0131362262, 0.0802817951 }, { 0.0193732295, 0.0151325789,
        0.0887668094 }, { 0.0224468865, 0.0171991484, 0.0973274383 }, { 0.0257927373, 0.0193306298,
        0.105929835 }, { 0.0294324251, 0.0215030771, 0.114621328 }, { 0.0333852235, 0.0237024271, 0.123397286
        }, { 0.0376684211, 0.0259207864, 0.132232108 }, { 0.0422525554, 0.0281385015, 0.141140519 }, {
        0.0469146287, 0.0303236129, 0.150163867 }, { 0.0516437624, 0.0324736172, 0.159254277 }, {
        0.0564491009, 0.0345691867, 0.168413539 }, { 0.06133972, 0.0365900213, 0.177642172 }, { 0.066331262,
        0.0385036268, 0.186961588 }, { 0.0714289181, 0.0402939095, 0.196353558 }, { 0.076636756, 0.0419053329,
        0.205798788 }, { 0.0819620773, 0.0433278666, 0.215289113 }, { 0.0874113897, 0.0445561662, 0.224813479
        }, { 0.0929901526, 0.0455829503, 0.234357604 }, { 0.0987024972, 0.0464018731, 0.2439037 }, {
        0.104550936, 0.0470080541, 0.2534303 }, { 0.110536084, 0.0473986708, 0.262912235 }, { 0.116656423,
        0.047573592, 0.272320803 }, { 0.122908126, 0.0475360183, 0.28162417 }, { 0.129284984, 0.0472930838,
        0.290788012 }, { 0.13577845, 0.0468563678, 0.299776404 }, { 0.142377819, 0.0462422566, 0.30855291 }, {
        0.149072957, 0.0454676444, 0.317085139 }, { 0.155849711, 0.0445588056, 0.325338414 }, { 0.162688939,
        0.0435542881, 0.333276678 }, { 0.169575148, 0.0424893149, 0.340874188 }, { 0.176493202, 0.0414017089,
        0.348110606 }, { 0.183428775, 0.0403288858, 0.354971391 }, { 0.190367453, 0.0393088888, 0.361446945 },
        { 0.197297425, 0.0384001825, 0.367534629 }, { 0.204209298, 0.0376322609, 0.373237557 }, { 0.211095463,
```

0.0370296488, 0.378563264 }, { 0.217948648, 0.0366146049, 0.383522415 }, { 0.224762908, 0.0364049901,
0.388128944 }, { 0.231538148, 0.0364052511, 0.39240015 }, { 0.238272961, 0.0366209949, 0.396353388 },
{ 0.244966911, 0.0370545017, 0.400006615 }, { 0.251620354, 0.0377052832, 0.403377897 }, { 0.258234265,
0.0385706153, 0.406485031 }, { 0.264809649, 0.0396468666, 0.409345373 }, { 0.271346664, 0.0409215821,
0.411976086 }, { 0.277849829, 0.0423528741, 0.414392106 }, { 0.284321318, 0.0439325787, 0.416607861 },
{ 0.290763373, 0.0456437598, 0.418636756 }, { 0.297178251, 0.0474700293, 0.420491164 }, { 0.303568182,
0.0493958927, 0.422182449 }, { 0.309935342, 0.0514069729, 0.423720999 }, { 0.316281835, 0.0534901321,
0.425116277 }, { 0.322609671, 0.0556335178, 0.426376869 }, { 0.328920763, 0.0578265505, 0.427510546 },
{ 0.335216916, 0.0600598734, 0.42852432 }, { 0.341499828, 0.0623252772, 0.429424503 }, { 0.347771086,
0.06461561, 0.430216765 }, { 0.354032169, 0.0669246832, 0.430906186 }, { 0.360284449, 0.0692471753,
0.431497309 }, { 0.366529195, 0.0715785403, 0.431994185 }, { 0.372767575, 0.0739149211, 0.432400419 },
{ 0.379000659, 0.0762530701, 0.432719214 }, { 0.385228383, 0.0785914864, 0.432954973 }, { 0.391452659,
0.0809267058, 0.433108763 }, { 0.397674379, 0.0832568129, 0.433182647 }, { 0.403894278, 0.0855803445,
0.433178526 }, { 0.410113015, 0.0878961593, 0.433098056 }, { 0.416331169, 0.0902033992, 0.432942678 },
{ 0.422549249, 0.0925014543, 0.432713635 }, { 0.428767696, 0.0947899342, 0.432411996 }, { 0.434986885,
0.0970686417, 0.432038673 }, { 0.441207124, 0.099337551, 0.431594438 }, { 0.447428382, 0.101597079,
0.431080497 }, { 0.453650614, 0.103847716, 0.430497898 }, { 0.459874623, 0.106089165, 0.429845789 }, {
0.466100494, 0.108321923, 0.429124507 }, { 0.472328255, 0.110546584, 0.42833432 }, { 0.478557889,
0.112763831, 0.427475431 }, { 0.484789325, 0.11497443, 0.426547991 }, { 0.491022448, 0.117179219,
0.425552106 }, { 0.497257069, 0.119379132, 0.424487908 }, { 0.503492698, 0.121575414, 0.42335611 }, {
0.509729541, 0.123768654, 0.422155676 }, { 0.515967304, 0.125959947, 0.420886594 }, { 0.522205646,
0.128150439, 0.419548848 }, { 0.528444192, 0.130341324, 0.418142411 }, { 0.534682523, 0.132533845,
0.416667258 }, { 0.540920186, 0.134729286, 0.415123366 }, { 0.547156706, 0.136928959, 0.413510662 }, {
0.553391649, 0.139134147, 0.411828882 }, { 0.559624442, 0.141346265, 0.410078028 }, { 0.565854477,
0.143566769, 0.408258132 }, { 0.572081108, 0.14579715, 0.406369246 }, { 0.578303656, 0.148038934,
0.404411444 }, { 0.584521407, 0.150293679, 0.402384829 }, { 0.590733615, 0.152562977, 0.400289528 }, {
0.596939751, 0.154848232, 0.398124897 }, { 0.60313893, 0.157151161, 0.395891308 }, { 0.609330184,
0.159473549, 0.393589349 }, { 0.615512627, 0.161817111, 0.391219295 }, { 0.62168534, 0.164183582,
0.388781456 }, { 0.627847374, 0.166574724, 0.38627618 }, { 0.633997746, 0.168992314, 0.383703854 }, {
0.640135447, 0.17143815, 0.381064906 }, { 0.646259648, 0.173913876, 0.378358969 }, { 0.652369348,
0.176421271, 0.375586209 }, { 0.658463166, 0.178962399, 0.372748214 }, { 0.664539964, 0.181539111,
0.369845599 }, { 0.670598572, 0.184153268, 0.366879025 }, { 0.676637795, 0.186806728, 0.363849195 }, {
0.682656407, 0.189501352, 0.360756856 }, { 0.688653158, 0.192238994, 0.357602797 }, { 0.694626769,
0.1950215, 0.354387853 }, { 0.700575937, 0.197850703, 0.3511129 }, { 0.706499709, 0.200728196,
0.347776863 }, { 0.712396345, 0.203656029, 0.344382594 }, { 0.718264447, 0.206635993, 0.340931208 }, {
0.724102613, 0.209669834, 0.337423766 }, { 0.729909422, 0.21275927, 0.333861367 }, { 0.735683432,
0.215905976, 0.330245147 }, { 0.741423185, 0.219111589, 0.326576275 }, { 0.747127207, 0.222377697,
0.322855952 }, { 0.752794009, 0.225705837, 0.31908541 }, { 0.75842209, 0.229097492, 0.31526591 }, {
0.76400994, 0.232554083, 0.311398734 }, { 0.769556038, 0.236076967, 0.307485188 }, { 0.775058888,
0.239667435, 0.303526312 }, { 0.780517023, 0.24332672, 0.299522665 }, { 0.785928794, 0.247055968,
0.295476756 }, { 0.791292674, 0.250856232, 0.291389943 }, { 0.796607144, 0.254728485, 0.287263585 }, {
0.801870689, 0.25867361, 0.283099033 }, { 0.807081807, 0.262692401, 0.278897629 }, { 0.812239008,
0.266785558, 0.274660698 }, { 0.817340818, 0.270953688, 0.270389545 }, { 0.822385784, 0.2751973,
0.266085445 }, { 0.827372474, 0.279516805, 0.261749643 }, { 0.832299481, 0.283912516, 0.257383341 }, {
0.837165425, 0.288384647, 0.2529877 }, { 0.841968959, 0.292933312, 0.248563825 }, { 0.846708768,
0.297558528, 0.244112767 }, { 0.851383572, 0.302260213, 0.239635512 }, { 0.85599213, 0.307038188,
0.235132978 }, { 0.860533241, 0.311892183, 0.230606009 }, { 0.865005747, 0.316821833, 0.226055368 }, {
0.869408534, 0.321826685, 0.221481734 }, { 0.87374053, 0.326906201, 0.216885699 }, { 0.878000715,
0.33205976, 0.212267762 }, { 0.882188112, 0.337286663, 0.207628326 }, { 0.886301795, 0.342586137,
0.202967696 }, { 0.890340885, 0.34795734, 0.19828608 }, { 0.894304553, 0.353399363, 0.193583583 }, {
0.898192017, 0.35891124, 0.188860212 }, { 0.902002544, 0.364491949, 0.184115876 }, { 0.905735448,
0.370140419, 0.179350388 }, { 0.90939009, 0.375855533, 0.174563472 }, { 0.912965874, 0.381636138,
0.169754764 }, { 0.916462251, 0.387481044, 0.164923826 }, { 0.91987871, 0.393389034, 0.160070152 }, {
0.923214783, 0.399358867, 0.155193185 }, { 0.926470039, 0.405389282, 0.150292329 }, { 0.929644083,
0.411479007, 0.145366973 }, { 0.932736555, 0.417626756, 0.140416519 }, { 0.935747126, 0.423831237,
0.135440416 }, { 0.938675494, 0.430091162, 0.130438175 }, { 0.941521384, 0.436405243, 0.12540944 }, {
0.944284543, 0.442772199, 0.120354038 }, { 0.946964741, 0.449190757, 0.115272059 }, { 0.949561766,
0.455659658, 0.110163947 }, { 0.952075421, 0.462177656, 0.105030614 }, { 0.954505523, 0.468743522,
0.0998735931 }, { 0.956851903, 0.475356048, 0.0946952268 }, { 0.959114397, 0.482014044, 0.0894989073
}, { 0.96129285, 0.488716345, 0.0842893891 }, { 0.96338711, 0.495461806, 0.0790731907 }, {
0.965397031, 0.502249309, 0.0738591143 }, { 0.967322465, 0.509077761, 0.0686589199 }, { 0.969163264,
0.515946092, 0.0634881971 }, { 0.970919277, 0.522853259, 0.058367489 }, { 0.972590351, 0.529798246,
0.0533237243 }, { 0.974176327, 0.536780059, 0.048392009 }, { 0.975677038, 0.543797733, 0.0436177922 },
{ 0.977092313, 0.550850323, 0.0390500131 }, { 0.978421971, 0.557936911, 0.0349306227 }, { 0.979665824,
0.5650566, 0.0314091591 }, { 0.980823673, 0.572208516, 0.0285075931 }, { 0.981895311, 0.579391803,
0.0262497353 }, { 0.982880522, 0.586605627, 0.0246613416 }, { 0.983779081, 0.593849168, 0.0237702263
}, { 0.984590755, 0.601121626, 0.0236063833 }, { 0.985315301, 0.608422211, 0.0242021174 }, {
0.985952471, 0.615750147, 0.0255921853 }, { 0.986502013, 0.623104667, 0.0278139496 }, { 0.98696367,
0.630485011, 0.0309075459 }, { 0.987337182, 0.637890424, 0.0349160639 }, { 0.987622296, 0.645320152,
0.0398857472 }, { 0.987818759, 0.652773439, 0.0455808037 }, { 0.98792633, 0.660249526, 0.0517503867 },
{ 0.987944783, 0.667747641, 0.0583286889 }, { 0.98787391, 0.675267, 0.0652570167 }, { 0.987713535,
0.682806802, 0.072489233 }, { 0.987463516, 0.690366218, 0.0799897176 }, { 0.987123759, 0.697944391,
0.0877314215 }, { 0.986694229, 0.705540424, 0.0956941797 }, { 0.98617497, 0.713153375, 0.103863324 },
{ 0.985565739, 0.72078246, 0.112228756 }, { 0.984865203, 0.728427497, 0.120784651 }, { 0.984075129,
0.736086521, 0.129526579 }, { 0.983195992, 0.743758326, 0.138453063 }, { 0.982228463, 0.751441596,
0.147564573 }, { 0.981173457, 0.759134892, 0.156863224 }, { 0.980032178, 0.766836624, 0.166352544 }, {
0.978806183, 0.774545028, 0.176037298 }, { 0.977497453, 0.782258138, 0.185923357 }, { 0.976108474,
0.789973753, 0.196017589 }, { 0.974637842, 0.797691563, 0.206331925 }, { 0.973087939, 0.805409333,
0.216876839 }, { 0.971467822, 0.813121725, 0.227658046 }, { 0.969783146, 0.820825143, 0.238685942 }, {
0.968040817, 0.828515491, 0.249971582 }, { 0.966242589, 0.836190976, 0.261533898 }, { 0.964393924,
0.843848069, 0.273391112 }, { 0.962516656, 0.85147634, 0.285545675 }, { 0.960625545, 0.859068716,
0.298010219 }, { 0.958720088, 0.866624355, 0.310820466 }, { 0.956834075, 0.874128569, 0.323973947 }, {
0.954997177, 0.881568926, 0.337475479 }, { 0.953215092, 0.888942277, 0.351368713 }, { 0.951546225,
0.896225909, 0.365627005 }, { 0.950018481, 0.903409063, 0.380271225 }, { 0.948683391, 0.910472964,
0.395289169 }, { 0.947594362, 0.917399053, 0.410665194 }, { 0.946809163, 0.924168246, 0.426373236 }, {
0.946391536, 0.930760752, 0.442367495 }, { 0.946402951, 0.937158971, 0.458591507 }, { 0.946902568,
0.943347775, 0.474969778 }, { 0.947936825, 0.949317522, 0.491426053 }, { 0.94954483, 0.9550629,

```
0.507859649 }, { 0.951740304, 0.960586693, 0.524203026 }, { 0.954529281, 0.965895868, 0.540360752 }, {
0.957896053, 0.97100333, 0.55627509 }, { 0.96181202, 0.975924241, 0.571925382 }, { 0.966248822,
0.980678193, 0.587205773 }, { 0.971161622, 0.985282161, 0.60215433 }, { 0.976510983, 0.989753437,
0.616760413 }, { 0.982257307, 0.994108844, 0.631017009 }, { 0.988362068, 0.998364143, 0.644924005 } };
00062         private static double[,] PlasmaRGB = new double[256, 3] { { 0.0503832136, 0.0298028976,
0.527974883 }, { 0.0635363639, 0.0284259729, 0.533123681 }, { 0.0753531234, 0.0272063728, 0.538007001
}, { 0.0862217979, 0.0261253206, 0.542657691 }, { 0.0963786097, 0.0251650976, 0.547103487 }, {
0.105979704, 0.0243092436, 0.551367851 }, { 0.115123641, 0.02355625, 0.555467728 }, { 0.123902903,
0.0228781011, 0.55942348 }, { 0.13238072, 0.0222583774, 0.563250116 }, { 0.140603076, 0.0216866674,
0.566959485 }, { 0.148606527, 0.0211535876, 0.570561711 }, { 0.156420649, 0.0206507174, 0.574065446 },
{ 0.164069722, 0.0201705326, 0.577478074 }, { 0.171573925, 0.0197063415, 0.58080589 }, { 0.178950212,
0.0192522243, 0.584054243 }, { 0.186212958, 0.0188029767, 0.587227661 }, { 0.193374449, 0.0183540593,
0.590329954 }, { 0.20044526, 0.0179015512, 0.593364304 }, { 0.207434551, 0.0174421086, 0.596333341 },
{ 0.214350298, 0.0169729276, 0.599239207 }, { 0.22119675, 0.0164970484, 0.602083323 }, { 0.227982971,
0.0160071509, 0.604867403 }, { 0.234714537, 0.0155015065, 0.607592438 }, { 0.241396253, 0.0149791041,
0.610259089 }, { 0.248032377, 0.0144393586, 0.612867743 }, { 0.25462669, 0.0138820918, 0.615418537 },
{ 0.261182562, 0.0133075156, 0.617911385 }, { 0.267702993, 0.0127162163, 0.620345997 }, { 0.274190665,
0.0121091423, 0.622721903 }, { 0.280647969, 0.0114875915, 0.625038468 }, { 0.287076059, 0.0108554862,
0.627294975 }, { 0.293477695, 0.0102128849, 0.62949049 }, { 0.299855122, 0.00956079551, 0.631623923 },
{ 0.306209825, 0.00890185346, 0.633694102 }, { 0.312543124, 0.00823900704, 0.635699759 }, {
0.318856183, 0.00757551051, 0.637639537 }, { 0.325150025, 0.00691491734, 0.639512001 }, { 0.331425547,
0.00626107379, 0.641315649 }, { 0.337683446, 0.00561830889, 0.643048936 }, { 0.343924591,
0.0049905308, 0.644710195 }, { 0.350149699, 0.00438202557, 0.646297711 }, { 0.356359209,
0.00379781761, 0.647809772 }, { 0.362553473, 0.00324319591, 0.649244641 }, { 0.368732762,
0.00272370721, 0.650600561 }, { 0.37489727, 0.00224514897, 0.651875762 }, { 0.381047116,
0.00181356205, 0.653068467 }, { 0.387182639, 0.00143446923, 0.654176761 }, { 0.39330401,
0.00111388259, 0.655198755 }, { 0.399410821, 0.000859420809, 0.656132835 }, { 0.405502914,
0.000678091517, 0.656977276 }, { 0.411580082, 0.000577101735, 0.65773038 }, { 0.417642063,
0.000563847476, 0.658390492 }, { 0.423688549, 0.00064590278, 0.658956004 }, { 0.429719186,
0.000831008207, 0.659425363 }, { 0.435733575, 0.00112705875, 0.659797077 }, { 0.441732123,
0.00153984779, 0.660069009 }, { 0.4477136, 0.00207954744, 0.660240367 }, { 0.453677394, 0.00275470302,
0.660309966 }, { 0.459622938, 0.00357374415, 0.660276655 }, { 0.465549631, 0.00454518084, 0.660139383
}, { 0.471456847, 0.00567758762, 0.65989721 }, { 0.477343929, 0.00697958743, 0.659549311 }, {
0.483210198, 0.00845983494, 0.659094989 }, { 0.489054951, 0.0101269996, 0.658533677 }, { 0.494877466,
0.0119897486, 0.657864946 }, { 0.500677087, 0.014055064, 0.657087561 }, { 0.506454143, 0.0163333443,
0.656202294 }, { 0.512206035, 0.0188332232, 0.655209222 }, { 0.51793258, 0.0215631918, 0.654108545 },
{ 0.52363299, 0.0245316468, 0.652900629 }, { 0.529306474, 0.0277468735, 0.65158601 }, { 0.534952244,
0.03121703, 0.650165396 }, { 0.54056951, 0.034950131, 0.648639668 }, { 0.546157494, 0.0389540334,
0.647009884 }, { 0.551715423, 0.0431364795, 0.645277275 }, { 0.557242538, 0.0473307585, 0.64344325 },
{ 0.562738096, 0.0515448092, 0.641509389 }, { 0.568201372, 0.0557776706, 0.63947744 }, { 0.573631859,
0.0600281369, 0.637348841 }, { 0.579028682, 0.0642955547, 0.635126108 }, { 0.584391137, 0.0685790261,
0.632811608 }, { 0.589718606, 0.0728775875, 0.630407727 }, { 0.595010505, 0.0771902878, 0.627916992 },
{ 0.600266283, 0.0815161895, 0.625342058 }, { 0.605485428, 0.0858543713, 0.622685703 }, { 0.610667469,
0.0902039303, 0.619950811 }, { 0.615811974, 0.0945639838, 0.617140367 }, { 0.620918555, 0.0989336721,
0.61425744 }, { 0.625986869, 0.10331216, 0.611305174 }, { 0.631016615, 0.107698641, 0.608286774 }, {
0.636007543, 0.112092335, 0.605205491 }, { 0.640959444, 0.116492495, 0.602064611 }, { 0.645872158,
0.120898405, 0.598867442 }, { 0.650745571, 0.125309384, 0.5956173 }, { 0.655579615, 0.129724785,
0.592317494 }, { 0.660374266, 0.134143997, 0.588971318 }, { 0.665129493, 0.138566428, 0.585582301 }, {
0.669845385, 0.14299154, 0.582153572 }, { 0.67452206, 0.147418835, 0.578688247 }, { 0.679159664,
0.151847851, 0.575189431 }, { 0.683758384, 0.156278163, 0.571660158 }, { 0.68831844, 0.160709387,
0.56810338 }, { 0.692840088, 0.165141174, 0.564521958 }, { 0.697323615, 0.169573215, 0.560918659 }, {
0.701769334, 0.174005236, 0.557296144 }, { 0.70617759, 0.178437, 0.55365697 }, { 0.710548747,
0.182868306, 0.550003579 }, { 0.714883195, 0.187298986, 0.546338299 }, { 0.719181339, 0.191728906,
0.542663338 }, { 0.723443604, 0.196157962, 0.538980786 }, { 0.727670428, 0.200586086, 0.535292612 }, {
0.731862231, 0.205013174, 0.531600995 }, { 0.736019424, 0.209439071, 0.527908434 }, { 0.740142557,
0.213863965, 0.524215533 }, { 0.744232102, 0.218287899, 0.520523766 }, { 0.748288533, 0.222710942,
0.516834495 }, { 0.752312321, 0.227133187, 0.513148963 }, { 0.756303937, 0.231554749, 0.509468305 }, {
0.760263849, 0.235975765, 0.505793543 }, { 0.764192516, 0.240396394, 0.502125599 }, { 0.768090391,
0.244816813, 0.49846529 }, { 0.771957916, 0.24923722, 0.494813338 }, { 0.775795522, 0.253657797,
0.491170517 }, { 0.779603614, 0.258078397, 0.487539124 }, { 0.783382636, 0.262499662, 0.483917732 }, {
0.787132978, 0.266921859, 0.480306702 }, { 0.790855015, 0.271345267, 0.476706319 }, { 0.794549101,
0.275770179, 0.473116798 }, { 0.798215577, 0.280196901, 0.469538286 }, { 0.801854758, 0.28462575,
0.465970871 }, { 0.805466945, 0.289057057, 0.46241458 }, { 0.809052419, 0.293491117, 0.458869577 }, {
0.812611506, 0.297927865, 0.455337565 }, { 0.816144382, 0.30236813, 0.451816385 }, { 0.819651255,
0.306812282, 0.448305861 }, { 0.823132309, 0.311260703, 0.444805781 }, { 0.826587706, 0.315713782,
0.441315901 }, { 0.830017584, 0.320171913, 0.437835947 }, { 0.833422053, 0.324635499, 0.434365616 }, {
0.836801237, 0.329104836, 0.430905052 }, { 0.840155276, 0.333580106, 0.427454836 }, { 0.843484103,
0.338062109, 0.424013059 }, { 0.846787726, 0.342551272, 0.420579333 }, { 0.850066132, 0.347048028,
0.417153264 }, { 0.853319279, 0.351552815, 0.413734445 }, { 0.856547103, 0.356066072, 0.410322469 }, {
0.85974952, 0.360588229, 0.406916975 }, { 0.862926559, 0.365119408, 0.403518809 }, { 0.86607792,
0.369660446, 0.400126027 }, { 0.869203436, 0.374211795, 0.396738211 }, { 0.872302917, 0.37877391,
0.393354947 }, { 0.875376149, 0.383347243, 0.389975832 }, { 0.878422895, 0.387932249, 0.386600468 }, {
0.881442916, 0.392529339, 0.383228622 }, { 0.884435982, 0.397138877, 0.379860246 }, { 0.887401682,
0.401761511, 0.376494232 }, { 0.890339687, 0.406397694, 0.373130228 }, { 0.893249647, 0.411047871,
0.369767893 }, { 0.896131191, 0.415712489, 0.366406907 }, { 0.898983931, 0.420391986, 0.363046965 }, {
0.901807455, 0.425086807, 0.359687758 }, { 0.904601295, 0.429797442, 0.356328796 }, { 0.907364995,
0.434524335, 0.352969777 }, { 0.910098088, 0.439267908, 0.349610469 }, { 0.912800095, 0.444028574,
0.346250656 }, { 0.915470518, 0.448806744, 0.342890148 }, { 0.918108848, 0.453602818, 0.339528771 }, {
0.920714383, 0.45841742, 0.336165582 }, { 0.92328666, 0.463250828, 0.332800827 }, { 0.925825146,
0.468103387, 0.329434512 }, { 0.928329275, 0.472975465, 0.32606655 }, { 0.930798469, 0.47786742,
0.322696876 }, { 0.93323214, 0.482779603, 0.319325444 }, { 0.935629684, 0.487712357, 0.315952211 }, {
0.937990034, 0.492666544, 0.31257544 }, { 0.940312939, 0.497642038, 0.309196628 }, { 0.942597771,
0.502639147, 0.305815824 }, { 0.944843893, 0.507658169, 0.302433101 }, { 0.947050662, 0.51269939,
0.299048555 }, { 0.949217427, 0.517763087, 0.295662308 }, { 0.95134353, 0.522849522, 0.292274506 }, {
0.953427725, 0.52795955, 0.288883445 }, { 0.95546964, 0.533093083, 0.285490391 }, { 0.95746877,
0.538250172, 0.282096149 }, { 0.95942443, 0.543431038, 0.27870099 }, { 0.96133593, 0.54863589,
0.275305214 }, { 0.963202573, 0.553864931, 0.271909159 }, { 0.965023656, 0.559118349, 0.2685132 }, {
```

```
        0.96679847, 0.564396327, 0.265117752 }, { 0.968525639, 0.569699633, 0.261721488 }, { 0.970204593,
        0.57502827, 0.258325424 }, { 0.971835007, 0.580382015, 0.254931256 }, { 0.973416145, 0.585761012,
        0.251539615 }, { 0.974947262, 0.591165394, 0.2481512 }, { 0.976427606, 0.596595287, 0.244766775 }, {
        0.977856416, 0.602050811, 0.241387186 }, { 0.979232922, 0.607532077, 0.238013359 }, { 0.980556344,
        0.61303919, 0.234646316 }, { 0.98182589, 0.61857225, 0.231287178 }, { 0.983040742, 0.624131362,
        0.227937141 }, { 0.984198924, 0.629717516, 0.224595006 }, { 0.98530076, 0.635329876, 0.221264889 }, {
        0.986345421, 0.640968508, 0.217948456 }, { 0.987332067, 0.646633475, 0.214647532 }, { 0.988259846,
        0.652324832, 0.211364122 }, { 0.989127893, 0.65804263, 0.208100426 }, { 0.989935328, 0.663786914,
        0.204858855 }, { 0.990681261, 0.66955772, 0.201642049 }, { 0.991364787, 0.675355082, 0.1984529 }, {
        0.99198499, 0.681179025, 0.195294567 }, { 0.992540939, 0.687029567, 0.1921705 }, { 0.993031693,
        0.692906719, 0.189084459 }, { 0.993456302, 0.698810484, 0.186040537 }, { 0.993813802, 0.704740854,
        0.18304318 }, { 0.994103226, 0.710697814, 0.180097207 }, { 0.994323596, 0.716681336, 0.177207826 }, {
        0.994473934, 0.722691379, 0.174380656 }, { 0.99455326, 0.72872789, 0.171621733 }, { 0.994560594,
        0.734790799, 0.168937522 }, { 0.994494964, 0.74088002, 0.166334918 }, { 0.994355411, 0.746995448,
        0.163821243 }, { 0.994140989, 0.753136955, 0.161404226 }, { 0.993850778, 0.75930439, 0.159091984 }, {
        0.99348219, 0.765498551, 0.156890625 }, { 0.993033251, 0.771719833, 0.154807583 }, { 0.992505214,
        0.777966775, 0.152854862 }, { 0.99189727, 0.78423912, 0.151041581 }, { 0.99120868, 0.790536569,
        0.149376885 }, { 0.990438793, 0.796858775, 0.14786981 }, { 0.989587065, 0.803205337, 0.146529128 }, {
        0.988647741, 0.809578605, 0.145357284 }, { 0.987620557, 0.815977942, 0.144362644 }, { 0.986509366,
        0.82240062, 0.143556679 }, { 0.985314198, 0.82884598, 0.142945116 }, { 0.984031139, 0.83531536,
        0.142528388 }, { 0.98265282, 0.84181173, 0.142302653 }, { 0.981190389, 0.848328902, 0.142278607 }, {
        0.979643637, 0.854866468, 0.142453425 }, { 0.977994918, 0.861432314, 0.142808191 }, { 0.976264977,
        0.868015998, 0.143350944 }, { 0.974443038, 0.874622194, 0.144061156 }, { 0.972530009, 0.881250063,
        0.144922913 }, { 0.970532932, 0.887896125, 0.145918663 }, { 0.968443477, 0.894563989, 0.147014438 }, {
        0.966271225, 0.901249365, 0.148179639 }, { 0.964021057, 0.907950379, 0.149370428 }, { 0.961681481,
        0.914672479, 0.150520343 }, { 0.959275646, 0.921406537, 0.151566019 }, { 0.956808068, 0.928152065,
        0.152409489 }, { 0.954286813, 0.93490773, 0.152921158 }, { 0.951726083, 0.941670605, 0.152925363 }, {
        0.949150533, 0.9484349, 0.152177604 }, { 0.94660227, 0.95518986, 0.150327944 }, { 0.944151742,
        0.961916487, 0.146860789 }, { 0.94189612, 0.968589814, 0.140955606 }, { 0.940015097, 0.975158357,
        0.131325517 } };
00063         private static double[,] ViridisRGB = new double[256, 3] { { 0.26700401, 0.00487433,
        0.32941519 }, { 0.26851048, 0.00960483, 0.33542652 }, { 0.26994384, 0.01462494, 0.34137895 }, {
        0.27130489, 0.01994186, 0.34726862 }, { 0.27259384, 0.02556309, 0.35309303 }, { 0.27380934,
        0.03149748, 0.35885256 }, { 0.27495242, 0.03775181, 0.36454323 }, { 0.27602238, 0.04416723, 0.37016418
        }, { 0.2770184, 0.05034437, 0.37571452 }, { 0.27794143, 0.05632444, 0.38119074 }, { 0.27879067,
        0.06214536, 0.38659204 }, { 0.2795655, 0.06783587, 0.39191723 }, { 0.28026658, 0.07341724, 0.39716349
        }, { 0.28089358, 0.07890703, 0.40232944 }, { 0.28144581, 0.0843197, 0.40741404 }, { 0.28192358,
        0.08966622, 0.41241521 }, { 0.28232739, 0.09495545, 0.41733086 }, { 0.28265633, 0.10019576, 0.42216032
        }, { 0.28291049, 0.10539345, 0.42690202 }, { 0.28309095, 0.11055307, 0.43155375 }, { 0.28319704,
        0.11567966, 0.43611482 }, { 0.28322882, 0.12077701, 0.44058404 }, { 0.28318684, 0.12584799, 0.44496 },
        { 0.283072, 0.13089477, 0.44924127 }, { 0.28288389, 0.13592005, 0.45342734 }, { 0.28262297,
        0.14092556, 0.45751726 }, { 0.28229037, 0.14591233, 0.46150995 }, { 0.28188676, 0.15088147, 0.46540474
        }, { 0.28141228, 0.15583425, 0.46920128 }, { 0.28086773, 0.16077132, 0.47289909 }, { 0.28025468,
        0.16569272, 0.47649762 }, { 0.27957399, 0.17059884, 0.47999675 }, { 0.27882618, 0.1754902, 0.48339654
        }, { 0.27801236, 0.18036684, 0.48669702 }, { 0.27713437, 0.18522836, 0.48989831 }, { 0.27619376,
        0.19007447, 0.49300074 }, { 0.27519116, 0.1949054, 0.49600488 }, { 0.27412802, 0.19972086, 0.49891131
        }, { 0.27300596, 0.20452049, 0.50172076 }, { 0.27182812, 0.20930306, 0.50443413 }, { 0.27059473,
        0.21406899, 0.50705243 }, { 0.26930756, 0.21881782, 0.50957678 }, { 0.26796846, 0.22354911, 0.5120084
        }, { 0.26657984, 0.2282621, 0.5143487 }, { 0.2651445, 0.23295593, 0.5165993 }, { 0.2636632,
        0.23763078, 0.51876163 }, { 0.26213801, 0.24228619, 0.52083736 }, { 0.26057103, 0.2469217, 0.52282822
        }, { 0.25896451, 0.25153685, 0.52473609 }, { 0.25732244, 0.2561304, 0.52656332 }, { 0.25564519,
        0.26070284, 0.52831152 }, { 0.25393498, 0.26525384, 0.52998273 }, { 0.25219404, 0.26978306, 0.53157905
        }, { 0.25042462, 0.27429024, 0.53310261 }, { 0.24862899, 0.27877509, 0.53455561 }, { 0.2468114,
        0.28323662, 0.53594093 }, { 0.24497208, 0.28767547, 0.53726018 }, { 0.24311324, 0.29209154, 0.53851561
        }, { 0.24123708, 0.29648471, 0.53970946 }, { 0.23934575, 0.30085494, 0.54084398 }, { 0.23744138,
        0.30520222, 0.5419214 }, { 0.23552606, 0.30952657, 0.54294396 }, { 0.23360277, 0.31382773, 0.54391424
        }, { 0.2316735, 0.3181058, 0.54483444 }, { 0.22973926, 0.32236127, 0.54570633 }, { 0.22780192,
        0.32659432, 0.546532 }, { 0.2258633, 0.33080515, 0.54731353 }, { 0.22392515, 0.334994, 0.54805291 }, {
        0.22198915, 0.33916114, 0.54875211 }, { 0.22005691, 0.34330688, 0.54941304 }, { 0.21812995,
        0.34743154, 0.55003755 }, { 0.21620971, 0.35153548, 0.55062743 }, { 0.21429757, 0.35561907, 0.5511844
        }, { 0.21239477, 0.35968273, 0.55171011 }, { 0.2105031, 0.36372671, 0.55220646 }, { 0.20862342,
        0.36775151, 0.55267486 }, { 0.20675628, 0.37175775, 0.55311653 }, { 0.20490257, 0.37574589, 0.55353282
        }, { 0.20306309, 0.37971644, 0.55392505 }, { 0.20123854, 0.38366989, 0.55429441 }, { 0.1994295,
        0.38760678, 0.55464205 }, { 0.1976365, 0.39152762, 0.55496905 }, { 0.19585993, 0.39543297, 0.55527637
        }, { 0.19410009, 0.39932336, 0.55556494 }, { 0.19235719, 0.40319934, 0.55583559 }, { 0.19063135,
        0.40706148, 0.55608907 }, { 0.18892259, 0.41091033, 0.55632606 }, { 0.18723083, 0.41474645, 0.55654717
        }, { 0.18555593, 0.4185704, 0.55675292 }, { 0.18389763, 0.42238275, 0.55694377 }, { 0.18225561,
        0.42618405, 0.5571201 }, { 0.18062949, 0.42997486, 0.55728221 }, { 0.17901879, 0.43375572, 0.55743035
        }, { 0.17742298, 0.4375272, 0.55756466 }, { 0.17584148, 0.44128981, 0.55768526 }, { 0.17427363,
        0.4450441, 0.55779216 }, { 0.17271876, 0.4487906, 0.55788532 }, { 0.17117615, 0.4525298, 0.55796464 },
        { 0.16964573, 0.45626209, 0.55803034 }, { 0.16812641, 0.45998802, 0.55808199 }, { 0.1666171,
        0.46370813, 0.55811913 }, { 0.16511703, 0.4674229, 0.55814141 }, { 0.16362543, 0.47113278, 0.55814842
        }, { 0.16214155, 0.47483821, 0.55813967 }, { 0.16066467, 0.47853961, 0.55811466 }, { 0.15919413,
        0.4822374, 0.5580728 }, { 0.15772933, 0.48593197, 0.55801347 }, { 0.15626973, 0.4896237, 0.557936 }, {
        0.15481488, 0.49331293, 0.55783967 }, { 0.15336445, 0.49700003, 0.55772371 }, { 0.1519182, 0.50068529,
        0.55758733 }, { 0.15047605, 0.50436904, 0.55742968 }, { 0.14903918, 0.50805136, 0.5572505 }, {
        0.14760731, 0.51173263, 0.55704861 }, { 0.14618026, 0.51541316, 0.55682271 }, { 0.14475863,
        0.51909319, 0.55657181 }, { 0.14334327, 0.52277292, 0.55629491 }, { 0.14193527, 0.52645254, 0.55599097
        }, { 0.14053599, 0.53013219, 0.55565893 }, { 0.13914708, 0.53381201, 0.55529773 }, { 0.13777048,
        0.53749213, 0.55490625 }, { 0.1364085, 0.54117264, 0.55448339 }, { 0.13506561, 0.54485335, 0.55402906
        }, { 0.13374299, 0.54853458, 0.55354108 }, { 0.13244401, 0.55221637, 0.55301828 }, { 0.13117249,
        0.55589872, 0.55245948 }, { 0.1299327, 0.55958162, 0.55186354 }, { 0.12872938, 0.56326503, 0.55122927
        }, { 0.12756771, 0.56694891, 0.55055551 }, { 0.12645338, 0.57063316, 0.5498411 }, { 0.12539383,
        0.57431754, 0.54908564 }, { 0.12439474, 0.57800205, 0.5482874 }, { 0.12346281, 0.58168661, 0.54744498
        }, { 0.12260562, 0.58537105, 0.54655722 }, { 0.12183122, 0.58905521, 0.54562298 }, { 0.12114807,
        0.59273889, 0.54464114 }, { 0.12056501, 0.59642187, 0.54361058 }, { 0.12009154, 0.60010387, 0.54253043
        }, { 0.11973756, 0.60378459, 0.54139999 }, { 0.11951163, 0.60746388, 0.54021751 }, { 0.11942341,
```

```
0.61114146, 0.53898192 }, { 0.11948255, 0.61481702, 0.53769219 }, { 0.11969858, 0.61849025, 0.53634733
}, { 0.12008079, 0.62216081, 0.53494633 }, { 0.12063824, 0.62582833, 0.53348834 }, { 0.12137972,
0.62949242, 0.53197275 }, { 0.12231244, 0.63315277, 0.53039808 }, { 0.12344358, 0.63680899, 0.52876343
}, { 0.12477953, 0.64046069, 0.52706792 }, { 0.12632581, 0.64410744, 0.52531069 }, { 0.12808703,
0.64774881, 0.52349092 }, { 0.13006688, 0.65138436, 0.52160791 }, { 0.13226797, 0.65501363, 0.51966086
}, { 0.13469183, 0.65863619, 0.5176488 }, { 0.13733921, 0.66225157, 0.51557101 }, { 0.14020991,
0.66585927, 0.5134268 }, { 0.14330291, 0.66945881, 0.51121549 }, { 0.1466164, 0.67304968, 0.50893644
}, { 0.15014782, 0.67663139, 0.5065889 }, { 0.15389405, 0.68020343, 0.50417217 }, { 0.15785146,
0.68376525, 0.50168574 }, { 0.16201598, 0.68731632, 0.49912906 }, { 0.1663832, 0.69085611, 0.49650163
}, { 0.1709484, 0.69438405, 0.49380294 }, { 0.17570671, 0.6978996, 0.49103252 }, { 0.18065314,
0.70140222, 0.48818938 }, { 0.18578266, 0.70489133, 0.48527326 }, { 0.19109018, 0.70836635, 0.48228395
}, { 0.19657063, 0.71182668, 0.47922108 }, { 0.20221902, 0.71527175, 0.47608431 }, { 0.20803045,
0.71870095, 0.4728733 }, { 0.21400015, 0.72211371, 0.46958774 }, { 0.22012381, 0.72550945, 0.46622638
}, { 0.2263969, 0.72888753, 0.46278934 }, { 0.23281498, 0.73224735, 0.45927675 }, { 0.2393739,
0.73558828, 0.45568838 }, { 0.24606968, 0.73890972, 0.45202405 }, { 0.25289851, 0.74221104, 0.44828355
}, { 0.25985676, 0.74549162, 0.44446673 }, { 0.26694127, 0.74875084, 0.44057284 }, { 0.27414922,
0.75198807, 0.4366009 }, { 0.28147681, 0.75520266, 0.43255207 }, { 0.28892102, 0.75839399, 0.42842626
}, { 0.29647899, 0.76156142, 0.42422341 }, { 0.30414796, 0.76470433, 0.41994346 }, { 0.31192534,
0.76782207, 0.41558638 }, { 0.3198086, 0.77091403, 0.41115215 }, { 0.3277958, 0.77397953, 0.40664011
}, { 0.33588539, 0.7770179, 0.40204917 }, { 0.34407411, 0.78002855, 0.39738103 }, { 0.35235985,
0.78301086, 0.39263579 }, { 0.36074053, 0.78596419, 0.38781353 }, { 0.3692142, 0.78888793, 0.38291438
}, { 0.37777892, 0.79178146, 0.3779385 }, { 0.38643282, 0.79464415, 0.37288606 }, { 0.39517408,
0.79747541, 0.36775726 }, { 0.40400101, 0.80027461, 0.36255223 }, { 0.4129135, 0.80304099, 0.35726893
}, { 0.42190813, 0.80577412, 0.35191009 }, { 0.43098317, 0.80847343, 0.34647607 }, { 0.44013691,
0.81113836, 0.3409673 }, { 0.44936763, 0.81376835, 0.33538426 }, { 0.45867362, 0.81636288, 0.32972749
}, { 0.46805314, 0.81892143, 0.32399761 }, { 0.47750446, 0.82144351, 0.31819529 }, { 0.4870258,
0.82392862, 0.31232133 }, { 0.49661536, 0.82637633, 0.30637661 }, { 0.5062713, 0.82878621, 0.30036211
}, { 0.51599182, 0.83115784, 0.29427888 }, { 0.52577622, 0.83349064, 0.2881265 }, { 0.5356211,
0.83578452, 0.28190832 }, { 0.5455244, 0.83803918, 0.27562602 }, { 0.55548397, 0.84025437, 0.26928147
}, { 0.5654976, 0.8424299, 0.26287683 }, { 0.57556297, 0.84456561, 0.25641457 }, { 0.58567772,
0.84666139, 0.24989748 }, { 0.59583934, 0.84871722, 0.24332878 }, { 0.60604528, 0.8507331, 0.23671214
}, { 0.61629283, 0.85270912, 0.23005179 }, { 0.62657923, 0.85464543, 0.22335258 }, { 0.63690157,
0.85654226, 0.21662012 }, { 0.64725685, 0.85839991, 0.20986086 }, { 0.65764197, 0.86021878, 0.20308229
}, { 0.66805369, 0.86199932, 0.19629307 }, { 0.67848868, 0.86374211, 0.18950326 }, { 0.68894351,
0.86544779, 0.18272455 }, { 0.69941463, 0.86711711, 0.17597055 }, { 0.70989842, 0.86875092, 0.16925712
}, { 0.72039115, 0.87035015, 0.16260273 }, { 0.73088902, 0.87191584, 0.15602894 }, { 0.74138803,
0.87344918, 0.14956101 }, { 0.75188414, 0.87495143, 0.14322828 }, { 0.76237342, 0.87642392, 0.13706449
}, { 0.77285183, 0.87786808, 0.13110864 }, { 0.78331535, 0.87928545, 0.12540538 }, { 0.79375994,
0.88067763, 0.12000532 }, { 0.80418159, 0.88204632, 0.11496505 }, { 0.81457634, 0.88339329, 0.11034678
}, { 0.82494028, 0.88472036, 0.10621724 }, { 0.83526959, 0.88602943, 0.1026459 }, { 0.84556056,
0.88732243, 0.09970219 }, { 0.8558096, 0.88860134, 0.09745186 }, { 0.86601325, 0.88986815, 0.09595277
}, { 0.87616824, 0.89112487, 0.09525046 }, { 0.88627146, 0.89237353, 0.09537439 }, { 0.89632002,
0.89361614, 0.09633538 }, { 0.90631121, 0.89485467, 0.09812496 }, { 0.91624212, 0.89609127, 0.1007168
}, { 0.92610579, 0.89732977, 0.10407067 }, { 0.93590444, 0.8985704, 0.10813094 }, { 0.94563626,
0.899815, 0.11283773 }, { 0.95529972, 0.90106534, 0.11812832 }, { 0.96489353, 0.90232311, 0.12394051
}, { 0.97441665, 0.90358991, 0.13021494 }, { 0.98386829, 0.90486726, 0.13689671 }, { 0.99324789,
0.90615657, 0.1439362 } };
00064         private static double[,] CividisRGB = new double[256, 3] { { 0, 0.1262, 0.3015 }, { 0, 0.1292,
0.3077 }, { 0, 0.1321, 0.3142 }, { 0, 0.135, 0.3205 }, { 0, 0.1379, 0.3269 }, { 0, 0.1408, 0.3334 }, {
0, 0.1437, 0.34 }, { 0, 0.1465, 0.3467 }, { 0, 0.1492, 0.3537 }, { 0, 0.1519, 0.3606 }, { 0, 0.1546,
0.3676 }, { 0, 0.1574, 0.3746 }, { 0, 0.1601, 0.3817 }, { 0, 0.1629, 0.3888 }, { 0, 0.1657, 0.396 }, {
0, 0.1685, 0.4031 }, { 0, 0.1714, 0.4102 }, { 0, 0.1743, 0.4172 }, { 0, 0.1773, 0.4241 }, { 0, 0.1798,
0.4307 }, { 0, 0.1817, 0.4347 }, { 0, 0.1834, 0.4363 }, { 0, 0.1852, 0.4368 }, { 0, 0.1872, 0.4368 },
{ 0, 0.1901, 0.4365 }, { 0, 0.193, 0.4361 }, { 0, 0.1958, 0.4356 }, { 0, 0.1987, 0.4349 }, { 0,
0.2015, 0.4343 }, { 0, 0.2044, 0.4336 }, { 0, 0.2073, 0.4329 }, { 0.0055, 0.2101, 0.4322 }, { 0.0236,
0.213, 0.4314 }, { 0.0416, 0.2158, 0.4308 }, { 0.0576, 0.2187, 0.4301 }, { 0.071, 0.2215, 0.4293 }, {
0.0827, 0.2244, 0.4287 }, { 0.0932, 0.2272, 0.428 }, { 0.103, 0.23, 0.4274 }, { 0.112, 0.2329, 0.4268
}, { 0.1204, 0.2357, 0.4262 }, { 0.1283, 0.2385, 0.4256 }, { 0.1359, 0.2414, 0.4251 }, { 0.1431,
0.2442, 0.4245 }, { 0.15, 0.247, 0.4241 }, { 0.1566, 0.2498, 0.4236 }, { 0.163, 0.2526, 0.4232 }, {
0.1692, 0.2555, 0.4228 }, { 0.1752, 0.2583, 0.4224 }, { 0.1811, 0.2611, 0.422 }, { 0.1868, 0.2639,
0.4217 }, { 0.1923, 0.2667, 0.4214 }, { 0.1977, 0.2695, 0.4212 }, { 0.203, 0.2723, 0.4209 }, { 0.2082,
0.2751, 0.4207 }, { 0.2133, 0.278, 0.4205 }, { 0.2183, 0.2808, 0.4204 }, { 0.2232, 0.2836, 0.4203 }, {
0.2281, 0.2864, 0.4202 }, { 0.2328, 0.2892, 0.4201 }, { 0.2375, 0.292, 0.42 }, { 0.2421, 0.2948, 0.42
}, { 0.2466, 0.2976, 0.42 }, { 0.2511, 0.3004, 0.4201 }, { 0.2556, 0.3032, 0.4201 }, { 0.2599, 0.306,
0.4202 }, { 0.2643, 0.3088, 0.4203 }, { 0.2686, 0.3116, 0.4205 }, { 0.2728, 0.3144, 0.4206 }, { 0.277,
0.3172, 0.4208 }, { 0.2811, 0.32, 0.421 }, { 0.2853, 0.3228, 0.4212 }, { 0.2894, 0.3256, 0.4215 }, {
0.2934, 0.3284, 0.4218 }, { 0.2974, 0.3312, 0.4221 }, { 0.3014, 0.334, 0.4224 }, { 0.3054, 0.3368,
0.4227 }, { 0.3093, 0.3396, 0.4231 }, { 0.3132, 0.3424, 0.4236 }, { 0.317, 0.3453, 0.424 }, { 0.3209,
0.3481, 0.4244 }, { 0.3247, 0.3509, 0.4249 }, { 0.3285, 0.3537, 0.4254 }, { 0.3323, 0.3565, 0.4259 },
{ 0.3361, 0.3593, 0.4264 }, { 0.3398, 0.3622, 0.427 }, { 0.3435, 0.365, 0.4276 }, { 0.3472, 0.3678,
0.4282 }, { 0.3509, 0.3706, 0.4288 }, { 0.3546, 0.3734, 0.4294 }, { 0.3582, 0.3763, 0.4302 }, {
0.3619, 0.3791, 0.4308 }, { 0.3655, 0.3819, 0.4316 }, { 0.3691, 0.3848, 0.4322 }, { 0.3727, 0.3876,
0.4331 }, { 0.3763, 0.3904, 0.4338 }, { 0.3798, 0.3933, 0.4346 }, { 0.3834, 0.3961, 0.4355 }, {
0.3869, 0.399, 0.4364 }, { 0.3905, 0.4018, 0.4372 }, { 0.394, 0.4047, 0.4381 }, { 0.3975, 0.4075,
0.439 }, { 0.401, 0.4104, 0.44 }, { 0.4045, 0.4132, 0.4409 }, { 0.408, 0.4161, 0.4419 }, { 0.4114,
0.4189, 0.443 }, { 0.4149, 0.4218, 0.444 }, { 0.4183, 0.4247, 0.445 }, { 0.4218, 0.4275, 0.4462 }, {
0.4252, 0.4304, 0.4473 }, { 0.4286, 0.4333, 0.4485 }, { 0.432, 0.4362, 0.4496 }, { 0.4354, 0.439,
0.4508 }, { 0.4388, 0.4419, 0.4521 }, { 0.4422, 0.4448, 0.4534 }, { 0.4456, 0.4477, 0.4547 }, {
0.4489, 0.4506, 0.4561 }, { 0.4523, 0.4535, 0.4575 }, { 0.4556, 0.4564, 0.4589 }, { 0.4589, 0.4593,
0.4604 }, { 0.4622, 0.4622, 0.462 }, { 0.4656, 0.4651, 0.4635 }, { 0.4689, 0.468, 0.465 }, { 0.4722,
0.4709, 0.4665 }, { 0.4756, 0.4738, 0.4679 }, { 0.479, 0.4767, 0.4691 }, { 0.4825, 0.4797, 0.4701 }, {
0.4861, 0.4826, 0.4707 }, { 0.4897, 0.4856, 0.4714 }, { 0.4934, 0.4886, 0.4719 }, { 0.4971, 0.4915,
0.4723 }, { 0.5008, 0.4945, 0.4727 }, { 0.5045, 0.4975, 0.473 }, { 0.5083, 0.5005, 0.4732 }, { 0.5121,
0.5035, 0.4734 }, { 0.5158, 0.5065, 0.4736 }, { 0.5196, 0.5095, 0.4737 }, { 0.5234, 0.5125, 0.4738 },
{ 0.5272, 0.5155, 0.4739 }, { 0.531, 0.5186, 0.4739 }, { 0.5349, 0.5216, 0.4738 }, { 0.5387, 0.5246,
0.4739 }, { 0.5425, 0.5277, 0.4738 }, { 0.5464, 0.5307, 0.4736 }, { 0.5502, 0.5338, 0.4735 }, {
0.5541, 0.5368, 0.4733 }, { 0.5579, 0.5399, 0.4732 }, { 0.5618, 0.543, 0.4729 }, { 0.5657, 0.5461,
```

```
0.4727 }, { 0.5696, 0.5491, 0.4723 }, { 0.5735, 0.5522, 0.472 }, { 0.5774, 0.5553, 0.4717 }, { 0.5813,
0.5584, 0.4714 }, { 0.5852, 0.5615, 0.4709 }, { 0.5892, 0.5646, 0.4705 }, { 0.5931, 0.5678, 0.4701 },
{ 0.597, 0.5709, 0.4696 }, { 0.601, 0.574, 0.4691 }, { 0.605, 0.5772, 0.4685 }, { 0.6089, 0.5803,
0.468 }, { 0.6129, 0.5835, 0.4673 }, { 0.6168, 0.5866, 0.4668 }, { 0.6208, 0.5898, 0.4662 }, { 0.6248,
0.5929, 0.4655 }, { 0.6288, 0.5961, 0.4649 }, { 0.6328, 0.5993, 0.4641 }, { 0.6368, 0.6025, 0.4632 },
{ 0.6408, 0.6057, 0.4625 }, { 0.6449, 0.6089, 0.4617 }, { 0.6489, 0.6121, 0.4609 }, { 0.6529, 0.6153,
0.46 }, { 0.657, 0.6185, 0.4591 }, { 0.661, 0.6217, 0.4583 }, { 0.6651, 0.625, 0.4573 }, { 0.6691,
0.6282, 0.4562 }, { 0.6732, 0.6315, 0.4553 }, { 0.6773, 0.6347, 0.4543 }, { 0.6813, 0.638, 0.4532 }, {
0.6854, 0.6412, 0.4521 }, { 0.6895, 0.6445, 0.4511 }, { 0.6936, 0.6478, 0.4499 }, { 0.6977, 0.6511,
0.4487 }, { 0.7018, 0.6544, 0.4475 }, { 0.706, 0.6577, 0.4463 }, { 0.7101, 0.661, 0.445 }, { 0.7142,
0.6643, 0.4437 }, { 0.7184, 0.6676, 0.4424 }, { 0.7225, 0.671, 0.4409 }, { 0.7267, 0.6743, 0.4396 }, {
0.7308, 0.6776, 0.4382 }, { 0.735, 0.681, 0.4368 }, { 0.7392, 0.6844, 0.4352 }, { 0.7434, 0.6877,
0.4338 }, { 0.7476, 0.6911, 0.4322 }, { 0.7518, 0.6945, 0.4307 }, { 0.756, 0.6979, 0.429 }, { 0.7602,
0.7013, 0.4273 }, { 0.7644, 0.7047, 0.4258 }, { 0.7686, 0.7081, 0.4241 }, { 0.7729, 0.7115, 0.4223 },
{ 0.7771, 0.715, 0.4205 }, { 0.7814, 0.7184, 0.4188 }, { 0.7856, 0.7218, 0.4168 }, { 0.7899, 0.7253,
0.415 }, { 0.7942, 0.7288, 0.4129 }, { 0.7985, 0.7322, 0.4111 }, { 0.8027, 0.7357, 0.409 }, { 0.807,
0.7392, 0.407 }, { 0.8114, 0.7427, 0.4049 }, { 0.8157, 0.7462, 0.4028 }, { 0.82, 0.7497, 0.4007 }, {
0.8243, 0.7532, 0.3984 }, { 0.8287, 0.7568, 0.3961 }, { 0.833, 0.7603, 0.3938 }, { 0.8374, 0.7639,
0.3915 }, { 0.8417, 0.7674, 0.3892 }, { 0.8461, 0.771, 0.3869 }, { 0.8505, 0.7745, 0.3843 }, { 0.8548,
0.7781, 0.3818 }, { 0.8592, 0.7817, 0.3793 }, { 0.8636, 0.7853, 0.3766 }, { 0.8681, 0.7889, 0.3739 },
{ 0.8725, 0.7926, 0.3712 }, { 0.8769, 0.7962, 0.3684 }, { 0.8813, 0.7998, 0.3657 }, { 0.8858, 0.8035,
0.3627 }, { 0.8902, 0.8071, 0.3599 }, { 0.8947, 0.8108, 0.3569 }, { 0.8992, 0.8145, 0.3538 }, {
0.9037, 0.8182, 0.3507 }, { 0.9082, 0.8219, 0.3474 }, { 0.9127, 0.8256, 0.3442 }, { 0.9172, 0.8293,
0.3409 }, { 0.9217, 0.833, 0.3374 }, { 0.9262, 0.8367, 0.334 }, { 0.9308, 0.8405, 0.3306 }, { 0.9353,
0.8442, 0.3268 }, { 0.9399, 0.848, 0.3232 }, { 0.9444, 0.8518, 0.3195 }, { 0.949, 0.8556, 0.3155 }, {
0.9536, 0.8593, 0.3116 }, { 0.9582, 0.8632, 0.3076 }, { 0.9628, 0.867, 0.3034 }, { 0.9674, 0.8708,
0.299 }, { 0.9721, 0.8746, 0.2947 }, { 0.9767, 0.8785, 0.2901 }, { 0.9814, 0.8823, 0.2856 }, { 0.986,
0.8862, 0.2807 }, { 0.9907, 0.8901, 0.2759 }, { 0.9954, 0.894, 0.2708 }, { 1, 0.8979, 0.2655 }, { 1,
0.9018, 0.26 }, { 1, 0.9057, 0.2593 }, { 1, 0.9094, 0.2634 }, { 1, 0.9131, 0.268 }, { 1, 0.9169,
0.2731 } };
00065         private static double[,] RocketRGB = new double[256, 3] { { 0.01060815, 0.01808215, 0.10018654
}, { 0.01428972, 0.02048237, 0.10374486 }, { 0.01831941, 0.0229766, 0.10738511 }, { 0.02275049,
0.02554464, 0.11108639 }, { 0.02759119, 0.02818316, 0.11483751 }, { 0.03285175, 0.03088792, 0.11863035
}, { 0.03853466, 0.03365771, 0.12245873 }, { 0.04447016, 0.03648425, 0.12631831 }, { 0.05032105,
0.03936808, 0.13020508 }, { 0.05611171, 0.04224835, 0.13411624 }, { 0.0618531, 0.04504866, 0.13804929
}, { 0.06755457, 0.04778179, 0.14200206 }, { 0.0732236, 0.05045047, 0.14597263 }, { 0.0788708,
0.05305461, 0.14995981 }, { 0.08450105, 0.05559631, 0.15396203 }, { 0.09011319, 0.05808059, 0.15797687
}, { 0.09572396, 0.06050127, 0.16200507 }, { 0.10132312, 0.06286782, 0.16604287 }, { 0.10692823,
0.06517224, 0.17009175 }, { 0.1125315, 0.06742194, 0.17414848 }, { 0.11813947, 0.06961499, 0.17821272
}, { 0.12375803, 0.07174938, 0.18228425 }, { 0.12938228, 0.07383015, 0.18636053 }, { 0.13501631,
0.07585609, 0.19044109 }, { 0.14066867, 0.0778224, 0.19452676 }, { 0.14633406, 0.07973393, 0.1986151
}, { 0.15201338, 0.08159108, 0.20270523 }, { 0.15770877, 0.08339312, 0.20679668 }, { 0.16342174,
0.0851396, 0.21088893 }, { 0.16915387, 0.08682996, 0.21498104 }, { 0.17489524, 0.08848235, 0.2190294
}, { 0.18065495, 0.09009031, 0.22303512 }, { 0.18643324, 0.09165431, 0.22699705 }, { 0.19223028,
0.09317479, 0.23091409 }, { 0.19804623, 0.09465217, 0.23478512 }, { 0.20388117, 0.09608689, 0.23860907
}, { 0.20973515, 0.09747934, 0.24238489 }, { 0.21560818, 0.09882993, 0.24611154 }, { 0.22150014,
0.10013944, 0.2497868 }, { 0.22741085, 0.10140876, 0.25340813 }, { 0.23334047, 0.10263737, 0.25697736
}, { 0.23928891, 0.10382562, 0.2604936 }, { 0.24525608, 0.10497384, 0.26395596 }, { 0.25124182,
0.10608236, 0.26736359 }, { 0.25724602, 0.10715148, 0.27071569 }, { 0.26326851, 0.1081815, 0.27401148
}, { 0.26930915, 0.1091727, 0.2772502 }, { 0.27536766, 0.11012568, 0.28043021 }, { 0.28144375,
0.11104133, 0.2835489 }, { 0.2875374, 0.11191896, 0.28660853 }, { 0.29364846, 0.11275876, 0.2896085 },
{ 0.29977678, 0.11356089, 0.29254823 }, { 0.30592213, 0.11432553, 0.29542718 }, { 0.31208435,
0.11505284, 0.29824485 }, { 0.31826327, 0.1157429, 0.30100076 }, { 0.32445869, 0.11639585, 0.30369448
}, { 0.33067031, 0.11701189, 0.30632563 }, { 0.33689808, 0.11759095, 0.3088938 }, { 0.34314168,
0.11813362, 0.31139721 }, { 0.34940101, 0.11863987, 0.3138355 }, { 0.355676, 0.11910909, 0.31620996 },
{ 0.36196644, 0.1195413, 0.31852037 }, { 0.36827206, 0.11993653, 0.32076656 }, { 0.37459292,
0.12029443, 0.32294825 }, { 0.38092887, 0.12061482, 0.32506528 }, { 0.38727975, 0.12089756, 0.3271175
}, { 0.39364518, 0.12114272, 0.32910494 }, { 0.40002537, 0.12134964, 0.33102734 }, { 0.40642019,
0.12151801, 0.33288464 }, { 0.41282936, 0.12164769, 0.33467689 }, { 0.41925278, 0.12173833, 0.33640407
}, { 0.42569057, 0.12178916, 0.33806605 }, { 0.43214263, 0.12179973, 0.33966284 }, { 0.43860848,
0.12177004, 0.34119475 }, { 0.44508855, 0.12169883, 0.34266151 }, { 0.45158266, 0.12158557, 0.34406324
}, { 0.45809049, 0.12142996, 0.34540024 }, { 0.46461238, 0.12123063, 0.34667231 }, { 0.47114798,
0.12098721, 0.34787978 }, { 0.47769736, 0.12069864, 0.34902273 }, { 0.48426077, 0.12036349, 0.35010104
}, { 0.49083761, 0.11998161, 0.35111537 }, { 0.49742847, 0.11955087, 0.35206533 }, { 0.50403286,
0.11907081, 0.35295152 }, { 0.51065109, 0.11853959, 0.35377385 }, { 0.51728314, 0.1179558, 0.35453252
}, { 0.52392883, 0.11731817, 0.35522789 }, { 0.53058853, 0.11662445, 0.35585982 }, { 0.53726173,
0.11587369, 0.35642903 }, { 0.54394898, 0.11506307, 0.35693521 }, { 0.5506426, 0.11420757, 0.35737863
}, { 0.55734473, 0.11330456, 0.35775059 }, { 0.56405586, 0.11235265, 0.35804813 }, { 0.57077365,
0.11135597, 0.35827146 }, { 0.5774991, 0.11031233, 0.35841679 }, { 0.58422945, 0.10922707, 0.35848469
}, { 0.59096382, 0.10810205, 0.35847347 }, { 0.59770215, 0.10693774, 0.35838029 }, { 0.60444226,
0.10573912, 0.35820487 }, { 0.61118304, 0.10450943, 0.35794557 }, { 0.61792306, 0.10325288, 0.35760108
}, { 0.62466162, 0.10197244, 0.35716891 }, { 0.63139686, 0.10067417, 0.35664819 }, { 0.63812122,
0.09938212, 0.35603757 }, { 0.64483795, 0.0980891, 0.35533555 }, { 0.65154562, 0.09680192, 0.35454107
}, { 0.65824241, 0.09552918, 0.3536529 }, { 0.66492652, 0.09428017, 0.3526697 }, { 0.67159578,
0.09306598, 0.35159077 }, { 0.67824099, 0.09192342, 0.3504148 }, { 0.684863, 0.09085633, 0.34914061 },
{ 0.69146268, 0.0898675, 0.34776864 }, { 0.69803757, 0.08897226, 0.3462986 }, { 0.70457834, 0.0882129,
0.34473046 }, { 0.71108138, 0.08761223, 0.3430635 }, { 0.7175507, 0.08716212, 0.34129974 }, {
0.72398193, 0.08688725, 0.33943958 }, { 0.73035829, 0.0868623, 0.33748452 }, { 0.73669146, 0.08704683,
0.33543669 }, { 0.74297501, 0.08747196, 0.33329799 }, { 0.74919318, 0.08820542, 0.33107204 }, {
0.75535825, 0.08919792, 0.32876184 }, { 0.76145589, 0.09050716, 0.32637117 }, { 0.76748424,
0.09213602, 0.32390525 }, { 0.77344838, 0.09405684, 0.32136808 }, { 0.77932641, 0.09634794, 0.31876642
}, { 0.78513609, 0.09892473, 0.31610488 }, { 0.79085854, 0.10184672, 0.313391 }, { 0.7965014,
0.10506637, 0.31063031 }, { 0.80205987, 0.10858333, 0.30783 }, { 0.80752799, 0.11239964, 0.30499738 },
{ 0.81291606, 0.11645784, 0.30213802 }, { 0.81820481, 0.12080606, 0.29926105 }, { 0.82341472,
0.12535343, 0.2963705 }, { 0.82852822, 0.13014118, 0.29347474 }, { 0.83355779, 0.13511035, 0.29057852
}, { 0.83850183, 0.14025098, 0.2876878 }, { 0.84335441, 0.14556683, 0.28480819 }, { 0.84813096,
0.15099892, 0.281943 }, { 0.85281737, 0.15657772, 0.27909826 }, { 0.85742602, 0.1622583, 0.27627462 },
```

```
      { 0.86196552, 0.16801239, 0.27346473 }, { 0.86641628, 0.17387796, 0.27070818 }, { 0.87079129,
      0.17982114, 0.26797378 }, { 0.87507281, 0.18587368, 0.26529697 }, { 0.87925878, 0.19203259, 0.26268136
      }, { 0.8833417, 0.19830556, 0.26014181 }, { 0.88731387, 0.20469941, 0.25769539 }, { 0.89116859,
      0.21121788, 0.2553592 }, { 0.89490337, 0.21785614, 0.25314362 }, { 0.8985026, 0.22463251, 0.25108745
      }, { 0.90197527, 0.23152063, 0.24918223 }, { 0.90530097, 0.23854541, 0.24748098 }, { 0.90848638,
      0.24568473, 0.24598324 }, { 0.911533, 0.25292623, 0.24470258 }, { 0.9144225, 0.26028902, 0.24369359 },
      { 0.91717106, 0.26773821, 0.24294137 }, { 0.91978131, 0.27526191, 0.24245973 }, { 0.92223947,
      0.28287251, 0.24229568 }, { 0.92456587, 0.29053388, 0.24242622 }, { 0.92676657, 0.29823282, 0.24285536
      }, { 0.92882964, 0.30598085, 0.24362274 }, { 0.93078135, 0.31373977, 0.24468803 }, { 0.93262051,
      0.3215093, 0.24606461 }, { 0.93435067, 0.32928362, 0.24775328 }, { 0.93599076, 0.33703942, 0.24972157
      }, { 0.93752831, 0.34479177, 0.25199928 }, { 0.93899289, 0.35250734, 0.25452808 }, { 0.94036561,
      0.36020899, 0.25734661 }, { 0.94167588, 0.36786594, 0.2603949 }, { 0.94291042, 0.37549479, 0.26369821
      }, { 0.94408513, 0.3830811, 0.26722004 }, { 0.94520419, 0.39062329, 0.27094924 }, { 0.94625977,
      0.39813168, 0.27489742 }, { 0.94727016, 0.4055909, 0.27902322 }, { 0.94823505, 0.41300424, 0.28332283
      }, { 0.94914549, 0.42038251, 0.28780969 }, { 0.95001704, 0.42771398, 0.29244728 }, { 0.95085121,
      0.43500005, 0.29722817 }, { 0.95165009, 0.44224144, 0.30214494 }, { 0.9524044, 0.44944853, 0.3072105
      }, { 0.95312556, 0.45661389, 0.31239776 }, { 0.95381595, 0.46373781, 0.31769923 }, { 0.95447591,
      0.47082238, 0.32310953 }, { 0.95510255, 0.47787236, 0.32862553 }, { 0.95569679, 0.48489115, 0.33421404
      }, { 0.95626788, 0.49187351, 0.33985601 }, { 0.95681685, 0.49882008, 0.34555431 }, { 0.9573439,
      0.50573243, 0.35130912 }, { 0.95784842, 0.51261283, 0.35711942 }, { 0.95833051, 0.51946267, 0.36298589
      }, { 0.95879054, 0.52628305, 0.36890904 }, { 0.95922872, 0.53307513, 0.3748895 }, { 0.95964538,
      0.53983991, 0.38092784 }, { 0.96004345, 0.54657593, 0.3870292 }, { 0.96042097, 0.55328624, 0.39319057
      }, { 0.96077819, 0.55997184, 0.39941173 }, { 0.9611152, 0.5666337, 0.40569343 }, { 0.96143273,
      0.57327231, 0.41203603 }, { 0.96173392, 0.57988594, 0.41844491 }, { 0.96201757, 0.58647675, 0.42491751
      }, { 0.96228344, 0.59304598, 0.43145271 }, { 0.96253168, 0.5995944, 0.43805131 }, { 0.96276513,
      0.60612062, 0.44471698 }, { 0.96298491, 0.6126247, 0.45145074 }, { 0.96318967, 0.61910879, 0.45824902
      }, { 0.96337949, 0.6255736, 0.46511271 }, { 0.96355923, 0.63201624, 0.47204746 }, { 0.96372785,
      0.63843852, 0.47905028 }, { 0.96388426, 0.64484214, 0.4861196 }, { 0.96403203, 0.65122535, 0.4932578
      }, { 0.96417332, 0.65758729, 0.50046894 }, { 0.9643063, 0.66393045, 0.5077467 }, { 0.96443322,
      0.67025402, 0.51509334 }, { 0.96455845, 0.67655564, 0.52251447 }, { 0.96467922, 0.68283846, 0.53000231
      }, { 0.96479861, 0.68910113, 0.53756026 }, { 0.96492035, 0.69534192, 0.5451917 }, { 0.96504223,
      0.7015636, 0.5528892 }, { 0.96516917, 0.70776351, 0.5606593 }, { 0.96530224, 0.71394212, 0.56849894 },
      { 0.96544032, 0.72010124, 0.57640375 }, { 0.96559206, 0.72623592, 0.58438387 }, { 0.96575293,
      0.73235058, 0.59242739 }, { 0.96592829, 0.73844258, 0.60053991 }, { 0.96612013, 0.74451182, 0.60871954
      }, { 0.96632832, 0.75055966, 0.61696136 }, { 0.96656022, 0.75658231, 0.62527295 }, { 0.96681185,
      0.76258381, 0.63364277 }, { 0.96709183, 0.76855969, 0.64207921 }, { 0.96739773, 0.77451297, 0.65057302
      }, { 0.96773482, 0.78044149, 0.65912731 }, { 0.96810471, 0.78634563, 0.66773889 }, { 0.96850919,
      0.79222565, 0.6764046 }, { 0.96893132, 0.79809112, 0.68512266 }, { 0.96935926, 0.80395415, 0.69383201
      }, { 0.9698028, 0.80981139, 0.70252255 }, { 0.97025511, 0.81566605, 0.71120296 }, { 0.97071849,
      0.82151775, 0.71987163 }, { 0.97120159, 0.82736371, 0.72851999 }, { 0.97169389, 0.83320847, 0.73716071
      }, { 0.97220061, 0.83905052, 0.74578903 }, { 0.97272597, 0.84488881, 0.75440141 }, { 0.97327085,
      0.85072354, 0.76299805 }, { 0.97383206, 0.85655639, 0.77158353 }, { 0.97441222, 0.86238689, 0.78015619
      }, { 0.97501782, 0.86821321, 0.78871034 }, { 0.97564391, 0.87403763, 0.79725261 }, { 0.97628674,
      0.87986189, 0.8057883 }, { 0.97696114, 0.88568129, 0.81430324 }, { 0.97765722, 0.89149971, 0.82280948
      }, { 0.97837585, 0.89731727, 0.83130786 }, { 0.97912374, 0.90313207, 0.83979337 }, { 0.979891,
      0.90894778, 0.84827858 }, { 0.98067764, 0.91476465, 0.85676611 }, { 0.98137749, 0.92061729, 0.86536915
      } };
00066         private static double[,] MakoRGB = new double[256, 3] { { 0.04503935, 0.01482344, 0.02092227
      }, { 0.04933018, 0.01709292, 0.02535719 }, { 0.05356262, 0.01950702, 0.03018802 }, { 0.05774337,
      0.02205989, 0.03545515 }, { 0.06188095, 0.02474764, 0.04115287 }, { 0.06598247, 0.0275665, 0.04691409
      }, { 0.07005374, 0.03051278, 0.05264306 }, { 0.07409947, 0.03358324, 0.05834631 }, { 0.07812339,
      0.03677446, 0.06403249 }, { 0.08212852, 0.0400833, 0.06970862 }, { 0.08611731, 0.04339148, 0.07538208
      }, { 0.09009161, 0.04664706, 0.08105568 }, { 0.09405308, 0.04985685, 0.08673591 }, { 0.09800301,
      0.05302279, 0.09242646 }, { 0.10194255, 0.05614641, 0.09813162 }, { 0.10587261, 0.05922941, 0.103854
      }, { 0.1097942, 0.06227277, 0.10959847 }, { 0.11370826, 0.06527747, 0.11536893 }, { 0.11761516,
      0.06824548, 0.12116393 }, { 0.12151575, 0.07117741, 0.12698763 }, { 0.12541095, 0.07407363, 0.1328442
      }, { 0.12930083, 0.07693611, 0.13873064 }, { 0.13317849, 0.07976988, 0.14465095 }, { 0.13701138,
      0.08259683, 0.15060265 }, { 0.14079223, 0.08542126, 0.15659379 }, { 0.14452486, 0.08824175, 0.16262484
      }, { 0.14820351, 0.09106304, 0.16869476 }, { 0.15183185, 0.09388372, 0.17480366 }, { 0.15540398,
      0.09670855, 0.18094993 }, { 0.15892417, 0.09953561, 0.18713384 }, { 0.16238588, 0.10236998, 0.19335329
      }, { 0.16579435, 0.10520905, 0.19960847 }, { 0.16914226, 0.10805832, 0.20589698 }, { 0.17243586,
      0.11091443, 0.21221911 }, { 0.17566717, 0.11378321, 0.21857219 }, { 0.17884322, 0.11666074, 0.2249565
      }, { 0.18195582, 0.11955283, 0.23136943 }, { 0.18501213, 0.12245547, 0.23781116 }, { 0.18800459,
      0.12537395, 0.24427914 }, { 0.19093944, 0.1283047, 0.25077369 }, { 0.19381092, 0.13125179, 0.25729255
      }, { 0.19662307, 0.13421303, 0.26383543 }, { 0.19937337, 0.13719028, 0.27040111 }, { 0.20206187,
      0.14018372, 0.27698891 }, { 0.20469116, 0.14319196, 0.28359861 }, { 0.20725547, 0.14621882, 0.29022775
      }, { 0.20976258, 0.14925954, 0.29687795 }, { 0.21220409, 0.15231929, 0.30354703 }, { 0.21458611,
      0.15539445, 0.31023563 }, { 0.21690827, 0.15848519, 0.31694355 }, { 0.21916481, 0.16159489, 0.32366939
      }, { 0.2213631, 0.16471913, 0.33041431 }, { 0.22349947, 0.1678599, 0.33717781 }, { 0.2255714,
      0.1710185, 0.34395925 }, { 0.22758415, 0.17419169, 0.35075983 }, { 0.22953569, 0.17738041, 0.35757941
      }, { 0.23142077, 0.18058733, 0.3644173 }, { 0.2332454, 0.18380872, 0.37127514 }, { 0.2350092,
      0.18704459, 0.3781528 }, { 0.23670785, 0.190297, 0.38504973 }, { 0.23834119, 0.19356547, 0.39196711 },
      { 0.23991189, 0.19684817, 0.39890581 }, { 0.24141903, 0.20014508, 0.4058667 }, { 0.24286214,
      0.20345642, 0.4128484 }, { 0.24423453, 0.20678459, 0.41985299 }, { 0.24554109, 0.21012669, 0.42688124
      }, { 0.2467815, 0.21348266, 0.43393244 }, { 0.24795393, 0.21685249, 0.4410088 }, { 0.24905614,
      0.22023618, 0.448113 }, { 0.25007383, 0.22365053, 0.45519562 }, { 0.25098926, 0.22710664, 0.46223892
      }, { 0.25179696, 0.23060342, 0.46925447 }, { 0.25249346, 0.23414353, 0.47623196 }, { 0.25307401,
      0.23772973, 0.48316271 }, { 0.25353152, 0.24136961, 0.49001976 }, { 0.25386167, 0.24506548, 0.49679407
      }, { 0.25406082, 0.2488164, 0.50348932 }, { 0.25412435, 0.25262843, 0.51007843 }, { 0.25404842,
      0.25650743, 0.51653282 }, { 0.25383134, 0.26044852, 0.52286845 }, { 0.2534705, 0.26446165, 0.52903422
      }, { 0.25296722, 0.2685428, 0.53503572 }, { 0.2523226, 0.27269346, 0.54085315 }, { 0.25153974,
      0.27691629, 0.54645752 }, { 0.25062402, 0.28120467, 0.55185939 }, { 0.24958205, 0.28556371, 0.55701246
      }, { 0.24842386, 0.28998148, 0.56194601 }, { 0.24715928, 0.29446327, 0.56660884 }, { 0.24580099,
      0.29899398, 0.57104399 }, { 0.24436202, 0.30357852, 0.57519929 }, { 0.24285591, 0.30819938, 0.57913247
      }, { 0.24129828, 0.31286235, 0.58278615 }, { 0.23970131, 0.3175495, 0.5862272 }, { 0.23807973,
      0.32226344, 0.58941872 }, { 0.23644557, 0.32699241, 0.59240198 }, { 0.2348113, 0.33173196, 0.59518282
      }, { 0.23318874, 0.33648036, 0.59775543 }, { 0.2315855, 0.34122763, 0.60016456 }, { 0.23001121,
```

```
0.34597357, 0.60240251 }, { 0.2284748, 0.35071512, 0.6044784 }, { 0.22698081, 0.35544612, 0.60642528
}, { 0.22553305, 0.36016515, 0.60825252 }, { 0.22413977, 0.36487341, 0.60994938 }, { 0.22280246,
0.36956728, 0.61154118 }, { 0.22152555, 0.37424409, 0.61304472 }, { 0.22030752, 0.37890437, 0.61446646
}, { 0.2191538, 0.38354668, 0.61581561 }, { 0.21806257, 0.38817169, 0.61709794 }, { 0.21703799,
0.39277882, 0.61831922 }, { 0.21607792, 0.39736958, 0.61948028 }, { 0.21518463, 0.40194196, 0.62059763
}, { 0.21435467, 0.40649717, 0.62167507 }, { 0.21358663, 0.41103579, 0.62271724 }, { 0.21288172,
0.41555771, 0.62373011 }, { 0.21223835, 0.42006355, 0.62471794 }, { 0.21165312, 0.42455441, 0.62568371
}, { 0.21112526, 0.42903064, 0.6266318 }, { 0.21065161, 0.43349321, 0.62756504 }, { 0.21023306,
0.43794288, 0.62848279 }, { 0.20985996, 0.44238227, 0.62938329 }, { 0.20951045, 0.44680966, 0.63030696
}, { 0.20916709, 0.45122981, 0.63124483 }, { 0.20882976, 0.45564335, 0.63219599 }, { 0.20849798,
0.46005094, 0.63315928 }, { 0.20817199, 0.46445309, 0.63413391 }, { 0.20785149, 0.46885041, 0.63511876
}, { 0.20753716, 0.47324327, 0.63611321 }, { 0.20722876, 0.47763224, 0.63711608 }, { 0.20692679,
0.48201774, 0.63812656 }, { 0.20663156, 0.48640018, 0.63914367 }, { 0.20634336, 0.49078002, 0.64016638
}, { 0.20606303, 0.49515755, 0.6411939 }, { 0.20578999, 0.49953341, 0.64222457 }, { 0.20552612,
0.50390766, 0.64325811 }, { 0.20527189, 0.50828072, 0.64429331 }, { 0.20502868, 0.51265277, 0.64532947
}, { 0.20479718, 0.51702417, 0.64636539 }, { 0.20457804, 0.52139527, 0.64739979 }, { 0.20437304,
0.52576622, 0.64843198 }, { 0.20418396, 0.53013715, 0.64946117 }, { 0.20401238, 0.53450825, 0.65048638
}, { 0.20385896, 0.53887991, 0.65150606 }, { 0.20372653, 0.54325208, 0.65251978 }, { 0.20361709,
0.5476249, 0.6535266 }, { 0.20353258, 0.55199854, 0.65452542 }, { 0.20347472, 0.55637318, 0.655515 },
{ 0.20344718, 0.56074869, 0.65649508 }, { 0.20345161, 0.56512531, 0.65746419 }, { 0.20349089,
0.56950304, 0.65842151 }, { 0.20356842, 0.57388184, 0.65936642 }, { 0.20368663, 0.57826181, 0.66029768
}, { 0.20384884, 0.58264293, 0.6612145 }, { 0.20405904, 0.58702506, 0.66211645 }, { 0.20431921,
0.59140842, 0.66300179 }, { 0.20463464, 0.59579264, 0.66387079 }, { 0.20500731, 0.60017798, 0.66472159
}, { 0.20544449, 0.60456387, 0.66555409 }, { 0.20596097, 0.60894927, 0.66636568 }, { 0.20654832,
0.61333521, 0.66715744 }, { 0.20721003, 0.61772167, 0.66792838 }, { 0.20795035, 0.62210845, 0.66867802
}, { 0.20877302, 0.62649546, 0.66940555 }, { 0.20968223, 0.63088252, 0.6701105 }, { 0.21068163,
0.63526951, 0.67079211 }, { 0.21177544, 0.63965621, 0.67145005 }, { 0.21298582, 0.64404072, 0.67208182
}, { 0.21430361, 0.64842404, 0.67268861 }, { 0.21572716, 0.65280655, 0.67326978 }, { 0.21726052,
0.65718791, 0.6738255 }, { 0.21890636, 0.66156803, 0.67435491 }, { 0.220668, 0.66594665, 0.67485792 },
{ 0.22255447, 0.67032297, 0.67533374 }, { 0.22458372, 0.67469531, 0.67578061 }, { 0.22673713,
0.67906542, 0.67620044 }, { 0.22901625, 0.6834332, 0.67659251 }, { 0.23142316, 0.68779836, 0.67695703
}, { 0.23395924, 0.69216072, 0.67729378 }, { 0.23663857, 0.69651881, 0.67760151 }, { 0.23946645,
0.70087194, 0.67788018 }, { 0.24242624, 0.70522162, 0.67813088 }, { 0.24549008, 0.70957083, 0.67835215
}, { 0.24863372, 0.71392166, 0.67854868 }, { 0.25187832, 0.71827158, 0.67872193 }, { 0.25524083,
0.72261873, 0.67887024 }, { 0.25870947, 0.72696469, 0.67898912 }, { 0.26229238, 0.73130855, 0.67907645
}, { 0.26604085, 0.73564353, 0.67914062 }, { 0.26993099, 0.73997282, 0.67917264 }, { 0.27397488,
0.74429484, 0.67917096 }, { 0.27822463, 0.74860229, 0.67914468 }, { 0.28264201, 0.75290034, 0.67907959
}, { 0.2873016, 0.75717817, 0.67899164 }, { 0.29215894, 0.76144162, 0.67886578 }, { 0.29729823,
0.76567816, 0.67871894 }, { 0.30268199, 0.76989232, 0.67853896 }, { 0.30835665, 0.77407636, 0.67833512
}, { 0.31435139, 0.77822478, 0.67811118 }, { 0.3206671, 0.78233575, 0.67786729 }, { 0.32733158,
0.78640315, 0.67761027 }, { 0.33437168, 0.79042043, 0.67734882 }, { 0.34182112, 0.79437948, 0.67709394
}, { 0.34968889, 0.79827511, 0.67685638 }, { 0.35799244, 0.80210037, 0.67664969 }, { 0.36675371,
0.80584651, 0.67649539 }, { 0.3759816, 0.80950627, 0.67641393 }, { 0.38566792, 0.81307432, 0.67642947
}, { 0.39579804, 0.81654592, 0.67656899 }, { 0.40634556, 0.81991799, 0.67686215 }, { 0.41730243,
0.82318339, 0.67735255 }, { 0.4285828, 0.82635051, 0.6780564 }, { 0.44012728, 0.82942353, 0.67900049
}, { 0.45189421, 0.83240398, 0.68021733 }, { 0.46378379, 0.83530763, 0.6817062 }, { 0.47573199,
0.83814472, 0.68347352 }, { 0.48769865, 0.84092197, 0.68552698 }, { 0.49962354, 0.84365379, 0.68783929
}, { 0.5114027, 0.8463718, 0.69029789 }, { 0.52301693, 0.84908401, 0.69288545 }, { 0.53447549,
0.85179048, 0.69561066 }, { 0.54578602, 0.8544913, 0.69848331 }, { 0.55695565, 0.85718723, 0.70150427
}, { 0.56798832, 0.85987893, 0.70468261 }, { 0.57888639, 0.86256715, 0.70802931 }, { 0.5896541,
0.8652532, 0.71154204 }, { 0.60028928, 0.86793835, 0.71523675 }, { 0.61079441, 0.87062438, 0.71910895
}, { 0.62116633, 0.87331311, 0.72317003 }, { 0.63140509, 0.87600675, 0.72741689 }, { 0.64150735,
0.87870746, 0.73185717 }, { 0.65147219, 0.8814179, 0.73648495 }, { 0.66129632, 0.8841403, 0.74130658
}, { 0.67097934, 0.88687758, 0.74631123 }, { 0.68051833, 0.88963189, 0.75150483 }, { 0.68991419,
0.89240612, 0.75687187 }, { 0.69916533, 0.89520211, 0.76241714 }, { 0.70827373, 0.89802257, 0.76812286
}, { 0.71723995, 0.90086891, 0.77399039 }, { 0.72606665, 0.90374337, 0.7800041 }, { 0.73475675,
0.90664718, 0.78615802 }, { 0.74331358, 0.90958151, 0.79244474 }, { 0.75174143, 0.91254787, 0.79884925
}, { 0.76004473, 0.91554656, 0.80536823 }, { 0.76827704, 0.91856549, 0.81196513 }, { 0.77647029,
0.921603, 0.81855729 }, { 0.78462009, 0.92466151, 0.82514119 }, { 0.79273542, 0.92773848, 0.83172131
}, { 0.8008109, 0.93083672, 0.83829355 }, { 0.80885107, 0.93395528, 0.84485982 }, { 0.81685878,
0.9370938, 0.85142101 }, { 0.82483206, 0.94025378, 0.8579751 }, { 0.83277661, 0.94343371, 0.86452477
}, { 0.84069127, 0.94663473, 0.87106853 }, { 0.84857662, 0.9498573, 0.8776059 }, { 0.8564431,
0.95309792, 0.88414253 }, { 0.86429066, 0.95635719, 0.89067759 }, { 0.87218969, 0.95960708, 0.89725384
} };
00067         private static double[,] TurboRGB = new double[256, 3] { { 0.18995, 0.07176, 0.23217 }, {
0.19483, 0.08339, 0.26149 }, { 0.19956, 0.09498, 0.29024 }, { 0.20415, 0.10652, 0.31844 }, { 0.2086,
0.11802, 0.34607 }, { 0.21291, 0.12947, 0.37314 }, { 0.21708, 0.14087, 0.39964 }, { 0.22111, 0.15223,
0.42558 }, { 0.225, 0.16354, 0.45096 }, { 0.22875, 0.17481, 0.47578 }, { 0.23236, 0.18603, 0.50004 },
{ 0.23582, 0.1972, 0.52373 }, { 0.23915, 0.20833, 0.54686 }, { 0.24234, 0.21941, 0.56942 }, { 0.24539,
0.23044, 0.59142 }, { 0.2483, 0.24143, 0.61286 }, { 0.25107, 0.25237, 0.63374 }, { 0.25369, 0.26327,
0.65406 }, { 0.25618, 0.27412, 0.67381 }, { 0.25853, 0.28492, 0.693 }, { 0.26074, 0.29568, 0.71162 },
{ 0.2628, 0.30639, 0.72968 }, { 0.26473, 0.31706, 0.74718 }, { 0.26652, 0.32768, 0.76412 }, { 0.26816,
0.33825, 0.7805 }, { 0.26967, 0.34878, 0.79631 }, { 0.27103, 0.35926, 0.81156 }, { 0.27226, 0.3697,
0.82624 }, { 0.27334, 0.38008, 0.84037 }, { 0.27429, 0.39043, 0.85393 }, { 0.27509, 0.40072, 0.86692
}, { 0.27576, 0.41097, 0.87936 }, { 0.27628, 0.42118, 0.89123 }, { 0.27667, 0.43134, 0.90254 }, {
0.27691, 0.44145, 0.91328 }, { 0.27701, 0.45152, 0.92347 }, { 0.27698, 0.46153, 0.93309 }, { 0.2768,
0.47151, 0.94214 }, { 0.27648, 0.48144, 0.95064 }, { 0.27603, 0.49132, 0.95857 }, { 0.27543, 0.50115,
0.96594 }, { 0.27469, 0.51094, 0.97275 }, { 0.27381, 0.52069, 0.97899 }, { 0.27273, 0.5304, 0.98461 },
{ 0.27106, 0.54015, 0.9893 }, { 0.26878, 0.54995, 0.99303 }, { 0.26592, 0.55979, 0.99583 }, { 0.26252,
0.56967, 0.99773 }, { 0.25862, 0.57958, 0.99876 }, { 0.25425, 0.5895, 0.99896 }, { 0.24946, 0.59943,
0.99835 }, { 0.24427, 0.60937, 0.99697 }, { 0.23874, 0.61931, 0.99485 }, { 0.23288, 0.62923, 0.99202
}, { 0.22676, 0.63913, 0.98851 }, { 0.22039, 0.64901, 0.98436 }, { 0.21382, 0.65886, 0.97959 }, {
0.20708, 0.66866, 0.97423 }, { 0.20021, 0.67842, 0.96833 }, { 0.19326, 0.68812, 0.9619 }, { 0.18625,
0.69775, 0.95498 }, { 0.17923, 0.70732, 0.94761 }, { 0.17223, 0.7168, 0.93981 }, { 0.16529, 0.7262,
0.93161 }, { 0.15844, 0.73551, 0.92305 }, { 0.15173, 0.74472, 0.91416 }, { 0.14519, 0.75381, 0.90496
}, { 0.13886, 0.76279, 0.8955 }, { 0.13278, 0.77165, 0.8858 }, { 0.12698, 0.78037, 0.8759 }, {
0.12151, 0.78896, 0.86581 }, { 0.11639, 0.7974, 0.85559 }, { 0.11167, 0.80569, 0.84525 }, { 0.10738,
```

```
      0.81381, 0.83484 }, { 0.10357, 0.82177, 0.82437 }, { 0.10026, 0.82955, 0.81389 }, { 0.0975, 0.83714,
      0.80342 }, { 0.09532, 0.84455, 0.79299 }, { 0.09377, 0.85175, 0.78264 }, { 0.09287, 0.85875, 0.7724 },
      { 0.09267, 0.86554, 0.7623 }, { 0.0932, 0.87211, 0.75237 }, { 0.09451, 0.87844, 0.74265 }, { 0.09662,
      0.88454, 0.73316 }, { 0.09958, 0.8904, 0.72393 }, { 0.10342, 0.896, 0.715 }, { 0.10815, 0.90142,
      0.70599 }, { 0.11374, 0.90673, 0.69651 }, { 0.12014, 0.91193, 0.6866 }, { 0.12733, 0.91701, 0.67627 },
      { 0.13526, 0.92197, 0.66556 }, { 0.14391, 0.9268, 0.65448 }, { 0.15323, 0.93151, 0.64308 }, { 0.16319,
      0.93609, 0.63137 }, { 0.17377, 0.94053, 0.61938 }, { 0.18491, 0.94484, 0.60713 }, { 0.19659, 0.94901,
      0.59466 }, { 0.20877, 0.95304, 0.58199 }, { 0.22142, 0.95692, 0.56914 }, { 0.23449, 0.96065, 0.55614
      }, { 0.24797, 0.96423, 0.54303 }, { 0.2618, 0.96765, 0.52981 }, { 0.27597, 0.97092, 0.51653 }, {
      0.29042, 0.97403, 0.50321 }, { 0.30513, 0.97697, 0.48987 }, { 0.32006, 0.97974, 0.47654 }, { 0.33517,
      0.98234, 0.46325 }, { 0.35043, 0.98477, 0.45002 }, { 0.36581, 0.98702, 0.43688 }, { 0.38127, 0.98909,
      0.42386 }, { 0.39678, 0.99098, 0.41098 }, { 0.41229, 0.99268, 0.39826 }, { 0.42778, 0.99419, 0.38575
      }, { 0.44321, 0.99551, 0.37345 }, { 0.45854, 0.99663, 0.3614 }, { 0.47375, 0.99755, 0.34963 }, {
      0.48879, 0.99828, 0.33816 }, { 0.50362, 0.99879, 0.32701 }, { 0.51822, 0.9991, 0.31622 }, { 0.53255,
      0.99919, 0.30581 }, { 0.54658, 0.99907, 0.29581 }, { 0.56026, 0.99873, 0.28623 }, { 0.57357, 0.99817,
      0.27712 }, { 0.58646, 0.99739, 0.26849 }, { 0.59891, 0.99638, 0.26038 }, { 0.61088, 0.99514, 0.2528 },
      { 0.62233, 0.99366, 0.24579 }, { 0.63323, 0.99195, 0.23937 }, { 0.64362, 0.98999, 0.23356 }, {
      0.65394, 0.98775, 0.22835 }, { 0.66428, 0.98524, 0.2237 }, { 0.67462, 0.98246, 0.2196 }, { 0.68494,
      0.97941, 0.21602 }, { 0.69525, 0.9761, 0.21294 }, { 0.70553, 0.97255, 0.21032 }, { 0.71577, 0.96875,
      0.20815 }, { 0.72596, 0.9647, 0.2064 }, { 0.7361, 0.96043, 0.20504 }, { 0.74617, 0.95593, 0.20406 }, {
      0.75617, 0.95121, 0.20343 }, { 0.76608, 0.94627, 0.20311 }, { 0.77591, 0.94113, 0.2031 }, { 0.78563,
      0.93579, 0.20336 }, { 0.79524, 0.93025, 0.20386 }, { 0.80473, 0.92452, 0.20459 }, { 0.8141, 0.91861,
      0.20552 }, { 0.82333, 0.91253, 0.20663 }, { 0.83241, 0.90627, 0.20788 }, { 0.84133, 0.89986, 0.20926
      }, { 0.8501, 0.89328, 0.21074 }, { 0.85868, 0.88655, 0.2123 }, { 0.86709, 0.87968, 0.21391 }, {
      0.8753, 0.87267, 0.21555 }, { 0.88331, 0.86553, 0.21719 }, { 0.89112, 0.85826, 0.2188 }, { 0.8987,
      0.85087, 0.22038 }, { 0.90605, 0.84337, 0.22188 }, { 0.91317, 0.83576, 0.22328 }, { 0.92004, 0.82806,
      0.22456 }, { 0.92666, 0.82025, 0.2257 }, { 0.93301, 0.81236, 0.22667 }, { 0.93909, 0.80439, 0.22744 },
      { 0.94489, 0.79634, 0.228 }, { 0.95039, 0.78823, 0.22831 }, { 0.9556, 0.78005, 0.22836 }, { 0.96049,
      0.77181, 0.22811 }, { 0.96507, 0.76352, 0.22754 }, { 0.96931, 0.75519, 0.22663 }, { 0.97323, 0.74682,
      0.22536 }, { 0.97679, 0.73842, 0.22369 }, { 0.98, 0.73, 0.22161 }, { 0.98289, 0.7214, 0.21918 }, {
      0.98549, 0.7125, 0.2165 }, { 0.98781, 0.7033, 0.21358 }, { 0.98986, 0.69382, 0.21043 }, { 0.99163,
      0.68408, 0.20706 }, { 0.99314, 0.67408, 0.20348 }, { 0.99438, 0.66386, 0.19971 }, { 0.99535, 0.65341,
      0.19577 }, { 0.99607, 0.64277, 0.19165 }, { 0.99654, 0.63193, 0.18738 }, { 0.99675, 0.62093, 0.18297
      }, { 0.99672, 0.60977, 0.17842 }, { 0.99644, 0.59846, 0.17376 }, { 0.99593, 0.58703, 0.16899 }, {
      0.99517, 0.57549, 0.16412 }, { 0.99419, 0.56386, 0.15918 }, { 0.99297, 0.55214, 0.15417 }, { 0.99153,
      0.54036, 0.1491 }, { 0.98987, 0.52854, 0.14398 }, { 0.98799, 0.51667, 0.13883 }, { 0.9859, 0.50479,
      0.13367 }, { 0.9836, 0.49291, 0.12849 }, { 0.98108, 0.48104, 0.12332 }, { 0.97837, 0.4692, 0.11817 },
      { 0.97545, 0.4574, 0.11305 }, { 0.97234, 0.44565, 0.10797 }, { 0.96904, 0.43399, 0.10294 }, { 0.96555,
      0.42241, 0.09798 }, { 0.96187, 0.41093, 0.0931 }, { 0.95801, 0.39958, 0.08831 }, { 0.95398, 0.38836,
      0.08362 }, { 0.94977, 0.37729, 0.07905 }, { 0.94538, 0.36638, 0.07461 }, { 0.94084, 0.35566, 0.07031
      }, { 0.93612, 0.34513, 0.06616 }, { 0.93125, 0.33482, 0.06218 }, { 0.92623, 0.32473, 0.05837 }, {
      0.92105, 0.31489, 0.05475 }, { 0.91572, 0.3053, 0.05134 }, { 0.91024, 0.29599, 0.04814 }, { 0.90463,
      0.28696, 0.04516 }, { 0.89888, 0.27824, 0.04243 }, { 0.89298, 0.26981, 0.03993 }, { 0.88691, 0.26152,
      0.03753 }, { 0.88066, 0.25334, 0.03521 }, { 0.87422, 0.24526, 0.03297 }, { 0.8676, 0.2373, 0.03082 },
      { 0.86079, 0.22945, 0.02875 }, { 0.8538, 0.2217, 0.02677 }, { 0.84662, 0.21407, 0.02487 }, { 0.83926,
      0.20654, 0.02305 }, { 0.83172, 0.19912, 0.02131 }, { 0.82399, 0.19182, 0.01966 }, { 0.81608, 0.18462,
      0.01809 }, { 0.80799, 0.17753, 0.0166 }, { 0.79971, 0.17055, 0.0152 }, { 0.79125, 0.16368, 0.01387 },
      { 0.7826, 0.15693, 0.01264 }, { 0.77377, 0.15028, 0.01148 }, { 0.76476, 0.14374, 0.01041 }, { 0.75556,
      0.13731, 0.00942 }, { 0.74617, 0.13098, 0.00851 }, { 0.73661, 0.12477, 0.00769 }, { 0.72686, 0.11867,
      0.00695 }, { 0.71692, 0.11268, 0.00629 }, { 0.7068, 0.1068, 0.00571 }, { 0.6965, 0.10102, 0.00522 }, {
      0.68602, 0.09536, 0.00481 }, { 0.67535, 0.0898, 0.00449 }, { 0.66449, 0.08436, 0.00424 }, { 0.65345,
      0.07902, 0.00408 }, { 0.64223, 0.0738, 0.00401 }, { 0.63082, 0.06868, 0.00401 }, { 0.61923, 0.06367,
      0.0041 }, { 0.60746, 0.05878, 0.00427 }, { 0.5955, 0.05399, 0.00453 }, { 0.58336, 0.04931, 0.00486 },
      { 0.57103, 0.04474, 0.00529 }, { 0.55852, 0.04028, 0.00579 }, { 0.54583, 0.03593, 0.00638 }, {
      0.53295, 0.03169, 0.00705 }, { 0.51989, 0.02756, 0.0078 }, { 0.50664, 0.02354, 0.00863 }, { 0.49321,
      0.01963, 0.00955 }, { 0.4796, 0.01583, 0.01055 } };
00068
00069 /// <summary>
00070 /// Magma gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
      href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00071 /// </summary>
00072         public static readonly GradientStops Magma = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(MagmaRGB[ind, 0],
      MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el * 0.05));
00073
00074 /// <summary>
00075 /// Inferno gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00076 /// </summary>
00077         public static readonly GradientStops Inferno = new GradientStops(from el in
      Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
      GradientStop(Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el * 0.05));
00078
00079 /// <summary>
00080 /// Plasma gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00081 /// </summary>
00082         public static readonly GradientStops Plasma = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(PlasmaRGB[ind, 0],
      PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el * 0.05));
00083
00084 /// <summary>
00085 /// Viridis gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00086 /// </summary>
00087         public static readonly GradientStops Viridis = new GradientStops(from el in
      Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
```

```
    GradientStop(Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el * 0.05));
00088
00089 /// <summary>
00090 /// Cividis gradient, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00091 /// </summary>
00092         public static readonly GradientStops Cividis = new GradientStops(from el in
    Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
    GradientStop(Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el * 0.05));
00093
00094 /// <summary>
00095 /// Rocket gradient, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00096 /// </summary>
00097         public static readonly GradientStops Rocket = new GradientStops(from el in Enumerable.Range(0,
    21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(RocketRGB[ind, 0],
    RocketRGB[ind, 1], RocketRGB[ind, 2]), el * 0.05));
00098
00099 /// <summary>
00100 /// Mako gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
    href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00101 /// </summary>
00102         public static readonly GradientStops Mako = new GradientStops(from el in Enumerable.Range(0,
    21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(MakoRGB[ind, 0],
    MakoRGB[ind, 1], MakoRGB[ind, 2]), el * 0.05));
00103
00104 /// <summary>
00105 /// Turbo gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
    href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00106 /// </summary>
00107         public static readonly GradientStops Turbo = new GradientStops(from el in Enumerable.Range(0,
    21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(TurboRGB[ind, 0],
    TurboRGB[ind, 1], TurboRGB[ind, 2]), el * 0.05));
00108
00109 /// <summary>
00110 /// Magma colour map, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00111 /// </summary>
00112 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00113 /// <returns>The colour at the corresponding position in the colour map.</returns>
00114         public static Colour MagmaColouring(double x)
00115         {
00116             x = Math.Max(Math.Min(x, 1), 0);
00117             int ind = (int)Math.Round(x * 255);
00118             return Colour.FromRgb(MagmaRGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]);
00119         }
00120
00121 /// <summary>
00122 /// Inferno colour map, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00123 /// </summary>
00124 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00125 /// <returns>The colour at the corresponding position in the colour map.</returns>
00126         public static Colour InfernoColouring(double x)
00127         {
00128             x = Math.Max(Math.Min(x, 1), 0);
00129             int ind = (int)Math.Round(x * 255);
00130             return Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]);
00131         }
00132
00133 /// <summary>
00134 /// Plasma colour map, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00135 /// </summary>
00136 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00137 /// <returns>The colour at the corresponding position in the colour map.</returns>
00138         public static Colour PlasmaColouring(double x)
00139         {
00140             x = Math.Max(Math.Min(x, 1), 0);
00141             int ind = (int)Math.Round(x * 255);
00142             return Colour.FromRgb(PlasmaRGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]);
00143         }
00144
00145 /// <summary>
00146 /// Viridis colour map, based on the homonymous colour scale included in the "viridis" R package
    (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00147 /// </summary>
00148 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00149 /// <returns>The colour at the corresponding position in the colour map.</returns>
00150         public static Colour ViridisColouring(double x)
00151         {
00152             x = Math.Max(Math.Min(x, 1), 0);
00153             int ind = (int)Math.Round(x * 255);
00154             return Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]);
00155         }
00156
00157 /// <summary>
```

```
00158 /// Cividis colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00159 /// </summary>
00160 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00161 /// <returns>The colour at the corresponding position in the colour map.</returns>
00162         public static Colour CividisColouring(double x)
00163         {
00164             x = Math.Max(Math.Min(x, 1), 0);
00165             int ind = (int)Math.Round(x * 255);
00166             return Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]);
00167         }
00168 
00169 /// <summary>
00170 /// Rocket colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00171 /// </summary>
00172 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00173 /// <returns>The colour at the corresponding position in the colour map.</returns>
00174         public static Colour RocketColouring(double x)
00175         {
00176             x = Math.Max(Math.Min(x, 1), 0);
00177             int ind = (int)Math.Round(x * 255);
00178             return Colour.FromRgb(RocketRGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]);
00179         }
00180 
00181 /// <summary>
00182 /// Mako colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00183 /// </summary>
00184 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00185 /// <returns>The colour at the corresponding position in the colour map.</returns>
00186         public static Colour MakoColouring(double x)
00187         {
00188             x = Math.Max(Math.Min(x, 1), 0);
00189             int ind = (int)Math.Round(x * 255);
00190             return Colour.FromRgb(MakoRGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]);
00191         }
00192 
00193 /// <summary>
00194 /// Turbo colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00195 /// </summary>
00196 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00197 /// <returns>The colour at the corresponding position in the colour map.</returns>
00198         public static Colour TurboColouring(double x)
00199         {
00200             x = Math.Max(Math.Min(x, 1), 0);
00201             int ind = (int)Math.Round(x * 255);
00202             return Colour.FromRgb(TurboRGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]);
00203         }
00204     }
00205 }
```

8.66 Graphics.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using VectSharp.Filters;
00021 
00022 namespace VectSharp
00023 {
00024     internal static class Utils
00025     {
00026         public static Point RotatePoint(Point point, double angle)
00027         {
00028             return new Point(point.X * Math.Cos(angle) - point.Y * Math.Sin(angle), point.X *
      Math.Sin(angle) + point.Y * Math.Cos(angle));
```

```
00029         }
00030     }
00031
00032 /// <summary>
00033 /// This interface should be implemented by classes intended to provide graphics output capability to
      a <see cref="Graphics"/> object.
00034 /// </summary>
00035     public interface IGraphicsContext
00036     {
00037 /// <summary>
00038 /// Width of the graphic surface.
00039 /// </summary>
00040         double Width { get; }
00041
00042 /// <summary>
00043 /// Height of the graphic surface.
00044 /// </summary>
00045         double Height { get; }
00046
00047 /// <summary>
00048 /// Save the current transform state (rotation, translation, scale).  This should be implemented as a
      LIFO stack.
00049 /// </summary>
00050         void Save();
00051
00052 /// <summary>
00053 /// Restore the previous transform state (rotation, translation, scale).  This should be implemented
      as a LIFO stack.
00054 /// </summary>
00055         void Restore();
00056
00057 /// <summary>
00058 /// Translate the coordinate system origin.
00059 /// </summary>
00060 /// <param name="x">The horizontal translation.</param>
00061 /// <param name="y">The vertical translation.</param>
00062         void Translate(double x, double y);
00063
00064 /// <summary>
00065 /// Rotate the coordinate system around the origin.
00066 /// </summary>
00067 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00068         void Rotate(double angle);
00069
00070 /// <summary>
00071 /// Scale the coordinate system with respect to the origin.
00072 /// </summary>
00073 /// <param name="scaleX">The horizontal scale.</param>
00074 /// <param name="scaleY">The vertical scale.</param>
00075         void Scale(double scaleX, double scaleY);
00076
00077 /// <summary>
00078 /// Transform the coordinate system with the specified transformation matrix [ [a, c, e], [b, d, f],
      [0, 0, 1] ].
00079 /// </summary>
00080 /// <param name="a">The first element of the first column.</param>
00081 /// <param name="b">The second element of the first column.</param>
00082 /// <param name="c">The first element of the second column.</param>
00083 /// <param name="d">The second element of the second column.</param>
00084 /// <param name="e">The first element of the third column.</param>
00085 /// <param name="f">The second element of the third column.</param>
00086         void Transform(double a, double b, double c, double d, double e, double f);
00087
00088 /// <summary>
00089 /// The current font.
00090 /// </summary>
00091         Font Font { get; set; }
00092
00093 /// <summary>
00094 /// The current text baseline.
00095 /// </summary>
00096         TextBaselines TextBaseline { get; set; }
00097
00098 /// <summary>
00099 /// Fill a text string using the current <see cref="Font"/> and <see cref="TextBaseline"/>.
00100 /// </summary>
00101 /// <param name="text">The string to draw.</param>
00102 /// <param name="x">The horizontal coordinate of the text origin.</param>
00103 /// <param name="y">The vertical coordinate of the text origin.</param>
00104         void FillText(string text, double x, double y);
00105
00106 /// <summary>
00107 /// Stroke the outline of a text string using the current <see cref="Font"/> and <see
      cref="TextBaseline"/>.
00108 /// </summary>
00109 /// <param name="text">The string to draw.</param>
00110 /// <param name="x">The horizontal coordinate of the text origin.</param>
```

```
00111 /// <param name="y">The vertical coordinate of the text origin.</param>
00112         void StrokeText(string text, double x, double y);
00113 
00114 /// <summary>
00115 /// Change the current point without drawing a line from the previous point.  If necessary, start a
      new figure.
00116 /// </summary>
00117 /// <param name="x">The horizontal coordinate of the point.</param>
00118 /// <param name="y">The vertical coordinate of the point.</param>
00119         void MoveTo(double x, double y);
00120 
00121 /// <summary>
00122 /// Draw a line from the previous point to the specified point.
00123 /// </summary>
00124 /// <param name="x">The horizontal coordinate of the point.</param>
00125 /// <param name="y">The vertical coordinate of the point.</param>
00126         void LineTo(double x, double y);
00127 
00128 /// <summary>
00129 /// Close the current figure.
00130 /// </summary>
00131         void Close();
00132 
00133 /// <summary>
00134 /// Stroke the current path using the current <see cref="StrokeStyle"/>, <see cref="LineWidth"/>, <see
      cref="LineCap"/>, <see cref="LineJoin"/> and <see cref="LineDash"/>.
00135 /// </summary>
00136         void Stroke();
00137 
00138 /// <summary>
00139 /// Set the current clipping path as the intersection of the previous clipping path and the current
      path.
00140 /// </summary>
00141         void SetClippingPath();
00142 
00143 /// <summary>
00144 /// Current brush used to fill paths.
00145 /// </summary>
00146         Brush FillStyle { get; }
00147 
00148 /// <summary>
00149 /// Set the current <see cref="FillStyle"/>.
00150 /// </summary>
00151 /// <param name="style">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00152         void SetFillStyle((int r, int g, int b, double a) style);
00153 
00154 /// <summary>
00155 /// Set the current <see cref="FillStyle"/>.
00156 /// </summary>
00157 /// <param name="style">The new fill style.</param>
00158         void SetFillStyle(Brush style);
00159 
00160 /// <summary>
00161 /// Current brush used to stroke paths.
00162 /// </summary>
00163         Brush StrokeStyle { get; }
00164 
00165 /// <summary>
00166 /// Set the current <see cref="StrokeStyle"/>.
00167 /// </summary>
00168 /// <param name="style">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00169         void SetStrokeStyle((int r, int g, int b, double a) style);
00170 
00171 /// <summary>
00172 /// Set the current <see cref="StrokeStyle"/>.
00173 /// </summary>
00174 /// <param name="style">The new stroke style.</param>
00175         void SetStrokeStyle(Brush style);
00176 
00177 /// <summary>
00178 /// Add to the current figure a cubic Bezier from the current point to a destination point, with two
      control points.
00179 /// </summary>
00180 /// <param name="p1X">The horizontal coordinate of the first control point.</param>
00181 /// <param name="p1Y">The vertical coordinate of the first control point.</param>
00182 /// <param name="p2X">The horizontal coordinate of the second control point.</param>
00183 /// <param name="p2Y">The vertical coordinate of the second control point.</param>
00184 /// <param name="p3X">The horizontal coordinate of the destination point.</param>
00185 /// <param name="p3Y">The vertical coordinate of the destination point.</param>
00186         void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double p3Y);
00187 
00188 /// <summary>
00189 /// Add a rectangle figure to the current path.
00190 /// </summary>
00191 /// <param name="x0">The horizontal coordinate of the top-left corner of the rectangle.</param>
```

```
00192 /// <param name="y0">The vertical coordinate of the top-left corner of the rectangle.</param>
00193 /// <param name="width">The width of corner of the rectangle.</param>
00194 /// <param name="height">The height of corner of the rectangle.</param>
00195         void Rectangle(double x0, double y0, double width, double height);
00196 
00197 /// <summary>
00198 /// Fill the current path using the current <see cref="FillStyle"/>.
00199 /// </summary>
00200 /// <param name="fillRule">The <see cref="FillRule"/> that determines which parts of the path are
      filled.</param>
00201         void Fill(FillRule fillRule);
00202 
00203 /// <summary>
00204 /// Current line width used to stroke paths.
00205 /// </summary>
00206         double LineWidth { get; set; }
00207 
00208 /// <summary>
00209 /// Current line cap used to stroke paths.
00210 /// </summary>
00211         LineCaps LineCap { set; }
00212 
00213 /// <summary>
00214 /// Current line join used to stroke paths.
00215 /// </summary>
00216         LineJoins LineJoin { set; }
00217 
00218 /// <summary>
00219 /// Set the current line dash pattern.
00220 /// </summary>
00221 /// <param name="dash">The line dash pattern.</param>
00222         void SetLineDash(LineDash dash);
00223 
00224 /// <summary>
00225 /// The current tag.  How this can be used depends on each implementation.
00226 /// </summary>
00227         string Tag { get; set; }
00228 
00229 /// <summary>
00230 /// Draw a raster image.
00231 /// </summary>
00232 /// <param name="sourceX">The horizontal coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00233 /// <param name="sourceY">The vertical coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00234 /// <param name="sourceWidth">The width of the rectangle delimiting the source area of the
      image.</param>
00235 /// <param name="sourceHeight">The height of the rectangle delimiting the source area of the
      image.</param>
00236 /// <param name="destinationX">The horizontal coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00237 /// <param name="destinationY">The vertical coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00238 /// <param name="destinationWidth">The width of the rectangle delimiting the destination area of the
      image.</param>
00239 /// <param name="destinationHeight">The height of the rectangle delimiting the destination area of the
      image.</param>
00240 /// <param name="image">The image to draw.</param>
00241         void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage
      image);
00242 
00243 /// <summary>
00244 /// Draws a <see cref="Graphics"/> object, applying the specified <paramref name="filter"/>.
00245 /// </summary>
00246 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00247 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00248         void DrawFilteredGraphics(Graphics graphics, IFilter filter);
00249     }
00250 
00251 /// <summary>
00252 /// The exception that is thrown when an unbalanced graphics state stack occurs.
00253 /// </summary>
00254     public class UnbalancedStackException :  Exception
00255     {
00256         internal UnbalancedStackException(int excessSave, int excessRestore) :  base("The graphics
      state stack is unbalanced!\nThere are " + excessSave + " calls to Graphics.Save() and " +
      excessRestore + " calls to Graphics.Restore() in excess!") { }
00257     }
00258 
00259 /// <summary>
00260 /// Represents a rule used to determine whether a point is inside or outside of a shape.
00261 /// </summary>
00262     public enum FillRule
00263     {
```

```
00264 /// <summary>
00265 /// A point is inside a shape if the perimeter of the shape is hit an odd number of times when moving
      from the point towards the outside.
00266 /// </summary>
00267         EvenOdd,
00268
00269 /// <summary>
00270 /// A point is inside a shape if its winding number is not zero.
00271 /// </summary>
00272         NonZeroWinding
00273     }
00274
00275 /// <summary>
00276 /// Represents an abstract drawing surface.
00277 /// </summary>
00278     public partial class Graphics
00279     {
00280 /// <summary>
00281 /// Determines how an unbalanced graphics state stack (which occurs if the number of calls to <see
      cref="Save"/> and <see cref="Restore"/> is not equal) will be treated.  The default is <see
      cref="UnbalancedStackActions.Throw"/>.
00282 /// </summary>
00283         public static UnbalancedStackActions UnbalancedStackAction { get; set; } =
      UnbalancedStackActions.Throw;
00284
00285         internal List<IGraphicsAction> Actions = new List<IGraphicsAction>();
00286
00287 /// <summary>
00288 /// The default fill rule.
00289 /// </summary>
00290         public FillRule DefaultFillRule { get; set; } = FillRule.NonZeroWinding;
00291
00292 /// <summary>
00293 /// Fill a <see cref="GraphicsPath"/>.
00294 /// </summary>
00295 /// <param name="path">The <see cref="GraphicsPath"/> to fill.</param>
00296 /// <param name="fillColour">The <see cref="Brush"/> with which to fill the <see
      cref="GraphicsPath"/>.</param>
00297 /// <param name="tag">A tag to identify the filled path.</param>
00298         public void FillPath(GraphicsPath path, Brush fillColour, string tag = null)
00299         {
00300             FillPath(path, fillColour, DefaultFillRule, tag);
00301         }
00302
00303 /// <summary>
00304 /// Fill a <see cref="GraphicsPath"/>.
00305 /// </summary>
00306 /// <param name="path">The <see cref="GraphicsPath"/> to fill.</param>
00307 /// <param name="fillColour">The <see cref="Brush"/> with which to fill the <see
      cref="GraphicsPath"/>.</param>
00308 /// <param name="fillRule">The <see cref="FillRule"/> that determines which parts of the path are
      filled.</param>
00309 /// <param name="tag">A tag to identify the filled path.</param>
00310         public void FillPath(GraphicsPath path, Brush fillColour, FillRule fillRule, string tag =
      null)
00311         {
00312             Actions.Add(new PathAction(path, fillColour, null, 0, LineCaps.Butt, LineJoins.Miter,
      LineDash.SolidLine, tag, fillRule, false));
00313         }
00314
00315 /// <summary>
00316 /// Determines whether unique tags should be used for graphics actions that create multiple objects
      (e.g.  drawing text).
00317 /// </summary>
00318         public bool UseUniqueTags { get; set; } = true;
00319
00320 /// <summary>
00321 /// Stroke a <see cref="GraphicsPath"/>.
00322 /// </summary>
00323 /// <param name="path">The <see cref="GraphicsPath"/> to stroke.</param>
00324 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the <see
      cref="GraphicsPath"/>.</param>
00325 /// <param name="lineWidth">The width of the line with which the path is stroked.</param>
00326 /// <param name="lineCap">The line cap to use to stroke the path.</param>
00327 /// <param name="lineJoin">The line join to use to stroke the path.</param>
00328 /// <param name="lineDash">The line dash to use to stroke the path.</param>
00329 /// <param name="tag">A tag to identify the stroked path.</param>
00330         public void StrokePath(GraphicsPath path, Brush strokeColour, double lineWidth = 1, LineCaps
      lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag
      = null)
00331         {
00332             Actions.Add(new PathAction(path, null, strokeColour, lineWidth, lineCap, lineJoin,
      lineDash ??  LineDash.SolidLine, tag, DefaultFillRule, false));
00333         }
00334
00335 /// <summary>
00336 /// Intersect the current clipping path with the specified <see cref="GraphicsPath"/>.
```

```
00337 /// </summary>
00338 /// <param name="path">The <see cref="GraphicsPath"/> to intersect with the current clipping
      path.</param>
00339 /// <param name="tag">A tag to identify the clipping path.</param>
00340         public void SetClippingPath(GraphicsPath path, string tag = null)
00341         {
00342             Actions.Add(new PathAction(path, null, null, 0, LineCaps.Butt, LineJoins.Miter,
      LineDash.SolidLine, tag, DefaultFillRule, true));
00343         }
00344 
00345 /// <summary>
00346 /// Intersect the current clipping path with the specified rectangle.
00347 /// </summary>
00348 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00349 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00350 /// <param name="width">The width of the rectangle.</param>
00351 /// <param name="height">The height of the rectangle.</param>
00352 /// <param name="tag">A tag to identify the clipping path.</param>
00353         public void SetClippingPath(double leftX, double topY, double width, double height, string tag
      = null)
00354         {
00355             SetClippingPath(new Point(leftX, topY), new Size(width, height), tag);
00356         }
00357 
00358 /// <summary>
00359 /// Intersect the current clipping path with the specified rectangle.
00360 /// </summary>
00361 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00362 /// <param name="size">The size of the rectangle.</param>
00363 /// <param name="tag">A tag to identify the clipping path.</param>
00364         public void SetClippingPath(Point topLeft, Size size, string tag = null)
00365         {
00366             Actions.Add(new PathAction(new GraphicsPath().MoveTo(topLeft).LineTo(topLeft.X +
      size.Width, topLeft.Y).LineTo(topLeft.X + size.Width, topLeft.Y + size.Height).LineTo(topLeft.X,
      topLeft.Y + size.Height).Close(), null, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine,
      tag, DefaultFillRule, true));
00367         }
00368 
00369 /// <summary>
00370 /// Rotate the coordinate system around the origin.
00371 /// </summary>
00372 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00373 /// <param name="tag">A tag to identify the transform.</param>
00374         public void Rotate(double angle, string tag = null)
00375         {
00376             Actions.Add(new TransformAction(angle, tag));
00377         }
00378 
00379 /// <summary>
00380 /// Rotate the coordinate system around a pivot point.
00381 /// </summary>
00382 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00383 /// <param name="pivot">The pivot around which the coordinate system is to be rotated.</param>
00384 /// <param name="tag">A tag to identify the transform.</param>
00385         public void RotateAt(double angle, Point pivot, string tag = null)
00386         {
00387             string tag1 = null;
00388             string tag2 = null;
00389             string tag3 = null;
00390 
00391             if (!string.IsNullOrEmpty(tag))
00392             {
00393                 if (UseUniqueTags)
00394                 {
00395                     tag1 = tag + "_@1";
00396                     tag2 = tag + "_@2";
00397                     tag3 = tag + "_@3";
00398                 }
00399                 else
00400                 {
00401                     tag1 = tag;
00402                     tag2 = tag;
00403                     tag3 = tag;
00404                 }
00405             }
00406 
00407             Actions.Add(new TransformAction(pivot, tag1));
00408             Actions.Add(new TransformAction(angle, tag2));
00409             Actions.Add(new TransformAction(new Point(-pivot.X, -pivot.Y), tag3));
00410         }
00411 
00412 
00413 /// <summary>
00414 /// Transform the coordinate system with the specified transformation matrix [ [a, c, e], [b, d, f],
      [0, 0, 1] ].
00415 /// </summary>
00416 /// <param name="a">The first element of the first column.</param>
```

```
00417 /// <param name="b">The second element of the first column.</param>
00418 /// <param name="c">The first element of the second column.</param>
00419 /// <param name="d">The second element of the second column.</param>
00420 /// <param name="e">The first element of the third column.</param>
00421 /// <param name="f">The second element of the third column.</param>
00422 /// <param name="tag">A tag to identify the transform.</param>
00423         public void Transform(double a, double b, double c, double d, double e, double f, string tag =
      null)
00424         {
00425             double[,] matrix = new double[,] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00426             Actions.Add(new TransformAction(matrix, tag));
00427         }
00428
00429 /// <summary>
00430 /// Translate the coordinate system origin.
00431 /// </summary>
00432 /// <param name="x">The horizontal translation.</param>
00433 /// <param name="y">The vertical translation.</param>
00434 /// <param name="tag">A tag to identify the transform.</param>
00435         public void Translate(double x, double y, string tag = null)
00436         {
00437             Actions.Add(new TransformAction(new Point(x, y), tag));
00438         }
00439
00440 /// <summary>
00441 /// Translate the coordinate system origin.
00442 /// </summary>
00443 /// <param name="delta">The new origin point.</param>
00444 /// <param name="tag">A tag to identify the transform.</param>
00445         public void Translate(Point delta, string tag = null)
00446         {
00447             Actions.Add(new TransformAction(delta, tag));
00448         }
00449
00450 /// <summary>
00451 /// Scale the coordinate system with respect to the origin.
00452 /// </summary>
00453 /// <param name="scaleX">The horizontal scale.</param>
00454 /// <param name="scaleY">The vertical scale.</param>
00455 /// <param name="tag">A tag to identify the transform.</param>
00456         public void Scale(double scaleX, double scaleY, string tag = null)
00457         {
00458             Actions.Add(new TransformAction(new Size(scaleX, scaleY), tag));
00459         }
00460
00461 /// <summary>
00462 /// Fill a rectangle.
00463 /// </summary>
00464 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00465 /// <param name="size">The size of the rectangle.</param>
00466 /// <param name="fillColour">The colour with which to fill the rectangle.</param>
00467 /// <param name="tag">A tag to identify the filled rectangle.</param>
00468         public void FillRectangle(Point topLeft, Size size, Brush fillColour, string tag = null)
00469         {
00470             Actions.Add(new RectangleAction(topLeft, size, fillColour, null, 0, LineCaps.Butt,
      LineJoins.Miter, LineDash.SolidLine, tag));
00471         }
00472
00473 /// <summary>
00474 /// Fill a rectangle.
00475 /// </summary>
00476 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00477 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00478 /// <param name="width">The width of the rectangle.</param>
00479 /// <param name="height">The height of the rectangle.</param>
00480 /// <param name="fillColour">The colour with which to fill the rectangle.</param>
00481 /// <param name="tag">A tag to identify the filled rectangle.</param>
00482         public void FillRectangle(double leftX, double topY, double width, double height, Brush
      fillColour, string tag = null)
00483         {
00484             Actions.Add(new RectangleAction(new Point(leftX, topY), new Size(width, height),
      fillColour, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00485         }
00486
00487 /// <summary>
00488 /// Stroke a rectangle.
00489 /// </summary>
00490 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00491 /// <param name="size">The size of the rectangle.</param>
00492 /// <param name="strokeColour">The colour with which to stroke the rectangle.</param>
00493 /// <param name="lineWidth">The width of the line with which the rectangle is stroked.</param>
00494 /// <param name="lineCap">The line cap to use to stroke the rectangle.</param>
00495 /// <param name="lineJoin">The line join to use to stroke the rectangle.</param>
00496 /// <param name="lineDash">The line dash to use to stroke the rectangle.</param>
00497 /// <param name="tag">A tag to identify the filled rectangle.</param>
00498         public void StrokeRectangle(Point topLeft, Size size, Brush strokeColour, double lineWidth =
      1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null,
```

```
    string tag = null)
00499         {
00500             Actions.Add(new RectangleAction(topLeft, size, null, strokeColour, lineWidth, lineCap,
    lineJoin, lineDash ??  LineDash.SolidLine, tag));
00501         }
00502
00503 /// <summary>
00504 /// Stroke a rectangle.
00505 /// </summary>
00506 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00507 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00508 /// <param name="width">The width of the rectangle.</param>
00509 /// <param name="height">The height of the rectangle.</param>
00510 /// <param name="strokeColour">The colour with which to stroke the rectangle.</param>
00511 /// <param name="lineWidth">The width of the line with which the rectangle is stroked.</param>
00512 /// <param name="lineCap">The line cap to use to stroke the rectangle.</param>
00513 /// <param name="lineJoin">The line join to use to stroke the rectangle.</param>
00514 /// <param name="lineDash">The line dash to use to stroke the rectangle.</param>
00515 /// <param name="tag">A tag to identify the filled rectangle.</param>
00516         public void StrokeRectangle(double leftX, double topY, double width, double height, Brush
    strokeColour, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin =
    LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00517         {
00518             Actions.Add(new RectangleAction(new Point(leftX, topY), new Size(width, height), null,
    strokeColour, lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00519         }
00520
00521 /// <summary>
00522 /// Draw a raster image.
00523 /// </summary>
00524 /// <param name="sourceX">The horizontal coordinate of the top-left corner of the rectangle delimiting
    the source area of the image.</param>
00525 /// <param name="sourceY">The vertical coordinate of the top-left corner of the rectangle delimiting
    the source area of the image.</param>
00526 /// <param name="sourceWidth">The width of the rectangle delimiting the source area of the
    image.</param>
00527 /// <param name="sourceHeight">The height of the rectangle delimiting the source area of the
    image.</param>
00528 /// <param name="destinationX">The horizontal coordinate of the top-left corner of the rectangle
    delimiting the destination area of the image.</param>
00529 /// <param name="destinationY">The vertical coordinate of the top-left corner of the rectangle
    delimiting the destination area of the image.</param>
00530 /// <param name="destinationWidth">The width of the rectangle delimiting the destination area of the
    image.</param>
00531 /// <param name="destinationHeight">The height of the rectangle delimiting the destination area of the
    image.</param>
00532 /// <param name="image">The image to draw.</param>
00533 /// <param name="tag">A tag to identify the drawn image.</param>
00534         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
    double destinationX, double destinationY, double destinationWidth, double destinationHeight,
    RasterImage image, string tag = null)
00535         {
00536             Actions.Add(new RasterImageAction(sourceX, sourceY, sourceWidth, sourceHeight,
    destinationX, destinationY, destinationWidth, destinationHeight, image, tag));
00537         }
00538
00539 /// <summary>
00540 /// Draw a raster image.
00541 /// </summary>
00542 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle delimiting the
    destination area of the image.</param>
00543 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle delimiting the
    destination area of the image.</param>
00544 /// <param name="image">The image to draw.</param>
00545 /// <param name="tag">A tag to identify the drawn image.</param>
00546         public void DrawRasterImage(double x, double y, RasterImage image, string tag = null)
00547         {
00548             DrawRasterImage(0, 0, image.Width, image.Height, x, y, image.Width, image.Height, image,
    tag);
00549         }
00550
00551 /// <summary>
00552 /// Draw a raster image.
00553 /// </summary>
00554 /// <param name="position">The the top-left corner of the rectangle delimiting the destination area of
    the image.</param>
00555 /// <param name="image">The image to draw.</param>
00556 /// <param name="tag">A tag to identify the drawn image.</param>
00557         public void DrawRasterImage(Point position, RasterImage image, string tag = null)
00558         {
00559             DrawRasterImage(0, 0, image.Width, image.Height, position.X, position.Y, image.Width,
    image.Height, image, tag);
00560         }
00561
00562 /// <summary>
00563 /// Draw a raster image.
00564 /// </summary>
```

```
00565 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00566 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00567 /// <param name="width">The width of the rectangle delimiting the destination area of the
      image.</param>
00568 /// <param name="height">The height of the rectangle delimiting the destination area of the
      image.</param>
00569 /// <param name="image">The image to draw.</param>
00570 /// <param name="tag">A tag to identify the drawn image.</param>
00571         public void DrawRasterImage(double x, double y, double width, double height, RasterImage
      image, string tag = null)
00572         {
00573             DrawRasterImage(0, 0, image.Width, image.Height, x, y, width, height, image, tag);
00574         }
00575
00576 /// <summary>
00577 /// Draw a raster image.
00578 /// </summary>
00579 /// <param name="position">The the top-left corner of the rectangle delimiting the destination area of
      the image.</param>
00580 /// <param name="size">The size of the rectangle delimiting the destination area of the image.</param>
00581 /// <param name="image">The image to draw.</param>
00582 /// <param name="tag">A tag to identify the drawn image.</param>
00583         public void DrawRasterImage(Point position, Size size, RasterImage image, string tag = null)
00584         {
00585             DrawRasterImage(0, 0, image.Width, image.Height, position.X, position.Y, size.Width,
      size.Height, image, tag);
00586         }
00587
00588 /// <summary>
00589 /// Save the current transform state (rotation, translation, scale).
00590 /// </summary>
00591         public void Save()
00592         {
00593             Actions.Add(new StateAction(StateAction.StateActionTypes.Save));
00594         }
00595
00596 /// <summary>
00597 /// Restore the previous transform state (rotation, translation scale).
00598 /// </summary>
00599         public void Restore()
00600         {
00601             Actions.Add(new StateAction(StateAction.StateActionTypes.Restore));
00602         }
00603
00604         private void FixGraphicsStateStack()
00605         {
00606             if (UnbalancedStackAction == UnbalancedStackActions.Ignore)
00607             {
00608                 return;
00609             }
00610
00611             int count = 0;
00612             List<int> toBeRemoved = new List<int>();
00613
00614             for (int i = 0; i < this.Actions.Count; i++)
00615             {
00616                 if (this.Actions[i] is StateAction st)
00617                 {
00618                     if (st.StateActionType == StateAction.StateActionTypes.Save)
00619                     {
00620                         count++;
00621                     }
00622                     else if (st.StateActionType == StateAction.StateActionTypes.Restore)
00623                     {
00624                         if (count == 0)
00625                         {
00626                             toBeRemoved.Add(i);
00627                         }
00628                         else
00629                         {
00630                             count--;
00631                         }
00632                     }
00633                 }
00634             }
00635
00636             if (UnbalancedStackAction == UnbalancedStackActions.Throw)
00637             {
00638                 if (count > 0 || toBeRemoved.Count > 0)
00639                 {
00640                     throw new UnbalancedStackException(count, toBeRemoved.Count);
00641                 }
00642             }
00643             else if (UnbalancedStackAction == UnbalancedStackActions.SilentlyFix)
00644             {
```

```
00645                 if (count > 0 || toBeRemoved.Count > 0)
00646                 {
00647                     for (int i = toBeRemoved.Count - 1; i >= 0; i--)
00648                     {
00649                         this.Actions.RemoveAt(toBeRemoved[i]);
00650                     }
00651 
00652                     for (int i = 0; i < count; i++)
00653                     {
00654                         this.Restore();
00655                     }
00656 
00657                     this.FixGraphicsStateStack();
00658                 }
00659             }
00660         }
00661 
00662 /// <summary>
00663 /// Copy the current graphics to an instance of a class implementing <see cref="IGraphicsContext"/>.
00664 /// </summary>
00665 /// <param name="destinationContext">The <see cref="IGraphicsContext"/> on which the graphics are to
      be copied.</param>
00666         public void CopyToIGraphicsContext(IGraphicsContext destinationContext)
00667         {
00668             this.FixGraphicsStateStack();
00669 
00670             for (int i = 0; i < this.Actions.Count; i++)
00671             {
00672                 if (this.Actions[i] is RectangleAction)
00673                 {
00674                     RectangleAction rec = this.Actions[i] as RectangleAction;
00675 
00676                     destinationContext.Tag = rec.Tag;
00677                     destinationContext.Rectangle(rec.TopLeft.X, rec.TopLeft.Y, rec.Size.Width,
      rec.Size.Height);
00678 
00679                     if (rec.Fill != null)
00680                     {
00681                         if (destinationContext.FillStyle != rec.Fill)
00682                         {
00683                             destinationContext.SetFillStyle(rec.Fill);
00684                         }
00685                         destinationContext.Fill(FillRule.EvenOdd);
00686                     }
00687                     else if (rec.Stroke != null)
00688                     {
00689                         if (destinationContext.StrokeStyle != rec.Stroke)
00690                         {
00691                             destinationContext.SetStrokeStyle(rec.Stroke);
00692                         }
00693                         if (destinationContext.LineWidth != rec.LineWidth)
00694                         {
00695                             destinationContext.LineWidth = rec.LineWidth;
00696                         }
00697                         destinationContext.SetLineDash(rec.LineDash);
00698                         destinationContext.LineCap = rec.LineCap;
00699                         destinationContext.LineJoin = rec.LineJoin;
00700 
00701                         destinationContext.Stroke();
00702                     }
00703                 }
00704                 else if (this.Actions[i] is PathAction)
00705                 {
00706                     PathAction pth = this.Actions[i] as PathAction;
00707 
00708                     destinationContext.Tag = pth.Tag;
00709 
00710                     for (int j = 0; j < pth.Path.Segments.Count; j++)
00711                     {
00712                         switch (pth.Path.Segments[j].Type)
00713                         {
00714                             case SegmentType.Move:
00715                                 destinationContext.MoveTo(pth.Path.Segments[j].Point.X,
      pth.Path.Segments[j].Point.Y);
00716                                 break;
00717                             case SegmentType.Line:
00718                                 destinationContext.LineTo(pth.Path.Segments[j].Point.X,
      pth.Path.Segments[j].Point.Y);
00719                                 break;
00720                             case SegmentType.CubicBezier:
00721                                 destinationContext.CubicBezierTo(pth.Path.Segments[j].Points[0].X,
      pth.Path.Segments[j].Points[0].Y, pth.Path.Segments[j].Points[1].X, pth.Path.Segments[j].Points[1].Y,
      pth.Path.Segments[j].Points[2].X, pth.Path.Segments[j].Points[2].Y);
00722                                 break;
00723                             case SegmentType.Arc:
00724                                 {
00725                                     ArcSegment seg = pth.Path.Segments[j] as ArcSegment;
```

```
00726                                   Segment[] segs = seg.ToBezierSegments();
00727                                   for (int k = 0; k < segs.Length; k++)
00728                                   {
00729                                       switch (segs[k].Type)
00730                                       {
00731                                           case SegmentType.Move:
00732                                               destinationContext.MoveTo(segs[k].Point.X,
      segs[k].Point.Y);
00733                                               break;
00734                                           case SegmentType.Line:
00735                                               destinationContext.LineTo(segs[k].Point.X,
      segs[k].Point.Y);
00736                                               break;
00737                                           case SegmentType.CubicBezier:
00738                                               destinationContext.CubicBezierTo(segs[k].Points[0].X,
      segs[k].Points[0].Y, segs[k].Points[1].X, segs[k].Points[1].Y, segs[k].Points[2].X,
      segs[k].Points[2].Y);
00739                                               break;
00740                                       }
00741                                   }
00742                               }
00743                               break;
00744                           case SegmentType.Close:
00745                               destinationContext.Close();
00746                               break;
00747                       }
00748                   }
00749
00750                   if (pth.IsClipping)
00751                   {
00752                       destinationContext.SetClippingPath();
00753                   }
00754                   else
00755                   {
00756                       if (pth.Fill != null)
00757                       {
00758                           if (destinationContext.FillStyle != pth.Fill)
00759                           {
00760                               destinationContext.SetFillStyle(pth.Fill);
00761                           }
00762                           destinationContext.Fill(pth.FillRule);
00763                       }
00764                       else if (pth.Stroke != null)
00765                       {
00766                           if (destinationContext.StrokeStyle != pth.Stroke)
00767                           {
00768                               destinationContext.SetStrokeStyle(pth.Stroke);
00769                           }
00770                           if (destinationContext.LineWidth != pth.LineWidth)
00771                           {
00772                               destinationContext.LineWidth = pth.LineWidth;
00773                           }
00774                           destinationContext.SetLineDash(pth.LineDash);
00775                           destinationContext.LineCap = pth.LineCap;
00776                           destinationContext.LineJoin = pth.LineJoin;
00777
00778                           destinationContext.Stroke();
00779                       }
00780                   }
00781               }
00782               else if (this.Actions[i] is TextAction)
00783               {
00784                   TextAction txt = this.Actions[i] as TextAction;
00785
00786                   destinationContext.Tag = txt.Tag;
00787                   if (destinationContext.TextBaseline != txt.TextBaseline)
00788                   {
00789                       destinationContext.TextBaseline = txt.TextBaseline;
00790                   }
00791                   destinationContext.Font = txt.Font;
00792
00793                   if (txt.Fill != null)
00794                   {
00795                       if (destinationContext.FillStyle != txt.Fill)
00796                       {
00797                           destinationContext.SetFillStyle(txt.Fill);
00798                       }
00799                       destinationContext.FillText(txt.Text, txt.Origin.X, txt.Origin.Y);
00800                   }
00801                   else if (txt.Stroke != null)
00802                   {
00803                       if (destinationContext.StrokeStyle != txt.Stroke)
00804                       {
00805                           destinationContext.SetStrokeStyle(txt.Stroke);
00806                       }
00807                       if (destinationContext.LineWidth != txt.LineWidth)
00808                       {
```

```
00809                           destinationContext.LineWidth = txt.LineWidth;
00810                       }
00811                       destinationContext.SetLineDash(txt.LineDash);
00812                       destinationContext.LineCap = txt.LineCap;
00813                       destinationContext.LineJoin = txt.LineJoin;
00814 
00815                       destinationContext.StrokeText(txt.Text, txt.Origin.X, txt.Origin.Y);
00816                   }
00817               }
00818               else if (this.Actions[i] is TransformAction)
00819               {
00820                   TransformAction trf = this.Actions[i] as TransformAction;
00821 
00822                   destinationContext.Tag = trf.Tag;
00823 
00824                   if (trf.Delta != null)
00825                   {
00826                       destinationContext.Translate(((Point)trf.Delta).X, ((Point)trf.Delta).Y);
00827                   }
00828                   else if (trf.Angle != null)
00829                   {
00830                       destinationContext.Rotate((double)trf.Angle);
00831                   }
00832                   else if (trf.Scale != null)
00833                   {
00834                       destinationContext.Scale(((Size)trf.Scale).Width, ((Size)trf.Scale).Height);
00835                   }
00836                   else if (trf.Matrix != null)
00837                   {
00838                       destinationContext.Transform(trf.Matrix[0, 0], trf.Matrix[1, 0], trf.Matrix[0,
      1], trf.Matrix[1, 1], trf.Matrix[0, 2], trf.Matrix[1, 2]);
00839                   }
00840               }
00841               else if (this.Actions[i] is StateAction)
00842               {
00843                   destinationContext.Tag = ((StateAction)this.Actions[i]).Tag;
00844 
00845                   if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
00846                   {
00847                       destinationContext.Save();
00848                   }
00849                   else
00850                   {
00851                       destinationContext.Restore();
00852                   }
00853               }
00854               else if (this.Actions[i] is RasterImageAction)
00855               {
00856                   RasterImageAction img = this.Actions[i] as RasterImageAction;
00857                   destinationContext.Tag = img.Tag;
00858                   destinationContext.DrawRasterImage(img.SourceX, img.SourceY, img.SourceWidth,
      img.SourceHeight, img.DestinationX, img.DestinationY, img.DestinationWidth, img.DestinationHeight,
      img.Image);
00859               }
00860               else if (this.Actions[i] is FilteredGraphicsAction)
00861               {
00862                   FilteredGraphicsAction fil = this.Actions[i] as FilteredGraphicsAction;
00863                   destinationContext.Tag = fil.Tag;
00864                   destinationContext.DrawFilteredGraphics(fil.Content, fil.Filter);
00865               }
00866           }
00867       }
00868 
00869 /// <summary>
00870 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object.
00871 /// </summary>
00872 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00873 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00874         public void DrawGraphics(Point origin, Graphics graphics)
00875         {
00876             this.Save();
00877             this.Translate(origin);
00878 
00879             graphics.FixGraphicsStateStack();
00880 
00881             this.Actions.AddRange(graphics.Actions);
00882 
00883             this.Restore();
00884         }
00885 
00886 /// <summary>
00887 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, prepending the
      supplied <paramref name="tag"/> to the tags contained in the <see cref="Graphics"/> object being
      drawn.
```

```
00888 /// </summary>
00889 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00890 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00891 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00892 /// <param name="tag">The tag to prepend to the tags contained in the <paramref name="graphics"/>
      object.</param>
00893         public void DrawGraphics(double originX, double originY, Graphics graphics, string tag)
00894         {
00895             this.DrawGraphics(new Point(originX, originY), graphics, tag);
00896         }
00897
00898 /// <summary>
00899 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, prepending the
      supplied <paramref name="tag"/> to the tags contained in the <see cref="Graphics"/> object being
      drawn.
00900 /// </summary>
00901 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00902 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00903 /// <param name="tag">The tag to prepend to the tags contained in the <paramref name="graphics"/>
      object.</param>
00904         public void DrawGraphics(Point origin, Graphics graphics, string tag)
00905         {
00906             this.Save();
00907
00908             if (!string.IsNullOrEmpty(tag) && UseUniqueTags)
00909             {
00910                 this.Translate(origin, tag + "@t");
00911             }
00912             else
00913             {
00914                 this.Translate(origin, tag);
00915             }
00916
00917             graphics.FixGraphicsStateStack();
00918
00919             foreach (IGraphicsAction action in graphics.Actions)
00920             {
00921                 IGraphicsAction clone = action.ShallowClone();
00922
00923                 if (clone is IPrintableAction print)
00924                 {
00925                     if (UseUniqueTags)
00926                     {
00927                         if (string.IsNullOrEmpty(print.Tag))
00928                         {
00929                             print.Tag = tag;
00930                         }
00931                         else
00932                         {
00933                             print.Tag = tag + "/" + print.Tag;
00934                         }
00935                     }
00936                 }
00937
00938                 this.Actions.Add(clone);
00939             }
00940
00941             this.Restore();
00942         }
00943
00944 /// <summary>
00945 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object.
00946 /// </summary>
00947 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00948 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00949 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00950         public void DrawGraphics(double originX, double originY, Graphics graphics)
00951         {
00952             this.DrawGraphics(new Point(originX, originY), graphics);
00953         }
00954
00955 /// <summary>
00956 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, applying the
      specified <paramref name="filter"/>.
00957 /// </summary>
00958 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00959 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
```

```
00960 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00961 /// <param name="tag">A tag to identify the filter.</param>
00962         public void DrawGraphics(Point origin, Graphics graphics, IFilter filter, string tag = null)
00963         {
00964             this.Save();
00965 
00966             if (!string.IsNullOrEmpty(tag) && UseUniqueTags)
00967             {
00968                 this.Translate(origin, tag + "@t");
00969             }
00970             else
00971             {
00972                 this.Translate(origin, tag);
00973             }
00974 
00975             Graphics clone = new Graphics();
00976             clone.Actions.AddRange(graphics.Actions);
00977             clone.FixGraphicsStateStack();
00978 
00979             this.Actions.Add(new FilteredGraphicsAction(clone, filter) { Tag = tag });
00980 
00981             this.Restore();
00982         }
00983 
00984 /// <summary>
00985 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, applying the
      specified <paramref name="filter"/>.
00986 /// </summary>
00987 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00988 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00989 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00990 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00991 /// <param name="tag">A tag to identify the filter.</param>
00992         public void DrawGraphics(double originX, double originY, Graphics graphics, IFilter filter,
      string tag = null)
00993         {
00994             this.DrawGraphics(new Point(originX, originY), graphics, filter, tag);
00995         }
00996 
00997 
00998         internal static Point Multiply(double[,] matrix, Point pt)
00999         {
01000             double x = matrix[0, 0] * pt.X + matrix[0, 1] * pt.Y + matrix[0, 2];
01001             double y = matrix[1, 0] * pt.X + matrix[1, 1] * pt.Y + matrix[1, 2];
01002             double z = matrix[2, 0] * pt.X + matrix[2, 1] * pt.Y + matrix[2, 2];
01003 
01004             return new Point(x / z, y / z);
01005         }
01006 
01007         internal static double[,] Multiply(double[,] m1, double[,] m2)
01008         {
01009             return new double[3, 3]
01010             {
01011                 { m1[0,0] * m2[0,0] + m1[0,1] * m2[1,0] + m1[0,2] *m2[2,0], m1[0,0] * m2[0,1] +
      m1[0,1] * m2[1,1] + m1[0,2] * m2[2,1], m1[0,0] * m2[0,2] + m1[0,1] * m2[1,2] + m1[0,2] * m2[2,2] },
01012                 { m1[1,0] * m2[0,0] + m1[1,1] * m2[1,0] + m1[1,2] *m2[2,0], m1[1,0] * m2[0,1] +
      m1[1,1] * m2[1,1] + m1[1,2] * m2[2,1], m1[1,0] * m2[0,2] + m1[1,1] * m2[1,2] + m1[1,2] * m2[2,2] },
01013                 { m1[2,0] * m2[0,0] + m1[2,1] * m2[1,0] + m1[2,2] *m2[2,0], m1[2,0] * m2[0,1] +
      m1[2,1] * m2[1,1] + m1[2,2] * m2[2,1], m1[2,0] * m2[0,2] + m1[2,1] * m2[1,2] + m1[2,2] * m2[2,2] }
01014             };
01015         }
01016 
01017         internal static double[,] TranslationMatrix(double dx, double dy)
01018         {
01019             return new double[3, 3]
01020             {
01021                 {1, 0, dx },
01022                 {0, 1, dy },
01023                 {0, 0, 1 }
01024             };
01025         }
01026 
01027         internal static double[,] ScaleMatrix(double sx, double sy)
01028         {
01029             return new double[3, 3]
01030             {
01031                 {sx, 0, 0 },
01032                 {0, sy, 0 },
01033                 {0, 0, 1 }
01034             };
01035         }
01036 
```

```
01037         internal static double[,] RotationMatrix(double theta)
01038         {
01039             return new double[3, 3]
01040             {
01041                 {Math.Cos(theta), -Math.Sin(theta), 0 },
01042                 {Math.Sin(theta), Math.Cos(theta), 0 },
01043                 {0, 0, 1 }
01044             };
01045         }
01046 
01047         internal static double[,] Invert(double[,] m)
01048         {
01049             double[,] tbr = new double[3, 3];
01050 
01051             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01052             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01053             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01054             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01055             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01056             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01057             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01058             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01059             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01060 
01061             return tbr;
01062         }
01063 
01064 /// <summary>
01065 /// Creates a new <see cref="Graphics"/> object in which all the graphics actions have been
      transformed using an arbitrary transformation function.  Raster images are replaced by grey
      rectangles.
01066 /// </summary>
01067 /// <param name="transformationFunction">An arbitrary transformation function.</param>
01068 /// <param name="linearisationResolution">The resolution that will be used to linearise curve
      segments.</param>
01069 /// <returns>A new <see cref="Graphics"/> object in which all graphics actions have been linearised
      and transformed using the <paramref name="transformationFunction"/>.</returns>
01070         public Graphics Transform(Func<Point, Point> transformationFunction, double
      linearisationResolution)
01071         {
01072             Graphics destinationGraphics = new Graphics();
01073 
01074             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01075             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01076 
01077             for (int i = 0; i < this.Actions.Count; i++)
01078             {
01079                 if (this.Actions[i] is RectangleAction)
01080                 {
01081                     RectangleAction rec = this.Actions[i] as RectangleAction;
01082 
01083                     GraphicsPath rectanglePath = new GraphicsPath();
01084 
01085                     Point pt1 = transformationFunction(Multiply(currMatrix, rec.TopLeft));
01086                     Point pt2 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X +
      rec.Size.Width, rec.TopLeft.Y)));
01087                     Point pt3 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X +
      rec.Size.Width, rec.TopLeft.Y + rec.Size.Height)));
01088                     Point pt4 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X,
      rec.TopLeft.Y + rec.Size.Height)));
01089 
01090                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01091 
01092                     if (rec.Fill != null)
01093                     {
01094                         destinationGraphics.FillPath(rectanglePath, rec.Fill, rec.Tag);
01095                     }
01096                     else if (rec.Stroke != null)
01097                     {
```

```
01098                         destinationGraphics.StrokePath(rectanglePath, rec.Stroke, rec.LineWidth,
      rec.LineCap, rec.LineJoin, rec.LineDash, rec.Tag);
01099                     }
01100                 }
01101                 else if (this.Actions[i] is PathAction)
01102                 {
01103                     PathAction pth = this.Actions[i] as PathAction;
01104 
01105                     GraphicsPath newPath = pth.Path.Linearise(linearisationResolution).Transform(pt =>
      transformationFunction(Multiply(currMatrix, pt)));
01106 
01107                     if (pth.IsClipping)
01108                     {
01109                         destinationGraphics.SetClippingPath(newPath);
01110                     }
01111                     else
01112                     {
01113                         if (pth.Fill != null)
01114                         {
01115                             destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01116                         }
01117                         else if (pth.Stroke != null)
01118                         {
01119                             destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
      pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01120                         }
01121                     }
01122                 }
01123                 else if (this.Actions[i] is TextAction)
01124                 {
01125                     TextAction txt = this.Actions[i] as TextAction;
01126 
01127                     GraphicsPath textPath = new GraphicsPath().AddText(txt.Origin, txt.Text, txt.Font,
      txt.TextBaseline).Linearise(linearisationResolution).Transform(pt =>
      transformationFunction(Multiply(currMatrix, pt)));
01128 
01129                     if (txt.Fill != null)
01130                     {
01131                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01132                     }
01133                     else if (txt.Stroke != null)
01134                     {
01135                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
      txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01136                     }
01137                 }
01138                 else if (this.Actions[i] is TransformAction)
01139                 {
01140                     TransformAction trf = this.Actions[i] as TransformAction;
01141 
01142                     if (trf.Delta != null)
01143                     {
01144                         currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01145                     }
01146                     else if (trf.Angle != null)
01147                     {
01148                         currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01149                     }
01150                     else if (trf.Scale != null)
01151                     {
01152                         currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01153                     }
01154                     else if (trf.Matrix != null)
01155                     {
01156                         currMatrix = Multiply(currMatrix, trf.Matrix);
01157                     }
01158                 }
01159                 else if (this.Actions[i] is StateAction)
01160                 {
01161                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01162                     {
01163                         transformMatrix.Push(currMatrix);
01164                     }
01165                     else
01166                     {
01167                         currMatrix = transformMatrix.Pop();
01168                     }
01169                 }
01170                 else if (this.Actions[i] is RasterImageAction)
01171                 {
01172                     RasterImageAction img = this.Actions[i] as RasterImageAction;
01173 
01174                     GraphicsPath rectanglePath = new GraphicsPath();
01175 
```

```
01176                     Point pt1 = transformationFunction(Multiply(currMatrix, new
      Point(img.DestinationX, img.DestinationY)));
01177                     Point pt2 = transformationFunction(Multiply(currMatrix, new Point(img.DestinationX
      + img.DestinationWidth, img.DestinationY)));
01178                     Point pt3 = transformationFunction(Multiply(currMatrix, new Point(img.DestinationX
      + img.DestinationWidth, img.DestinationY + img.DestinationHeight)));
01179                     Point pt4 = transformationFunction(Multiply(currMatrix, new
      Point(img.DestinationX, img.DestinationY + img.DestinationHeight)));
01180
01181                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01182
01183                     destinationGraphics.FillPath(rectanglePath, Colour.FromRgb(220, 220, 220),
      img.Tag);
01184                 }
01185             }
01186
01187             return destinationGraphics;
01188         }
01189
01190         internal static GraphicsPath ReduceMaximumLength(GraphicsPath path, double maxLength)
01191         {
01192             GraphicsPath shortLinearisedPath = new GraphicsPath();
01193
01194             // Square of the max length - so that we avoid the square roots.
01195             double maxLengthSq = maxLength * maxLength;
01196
01197             Point currPoint = new Point();
01198             Point startFigurePoint = new Point();
01199
01200             foreach (Segment seg in path.Segments)
01201             {
01202                 if (seg is MoveSegment mov)
01203                 {
01204                     shortLinearisedPath.MoveTo(mov.Point);
01205                     startFigurePoint = mov.Point;
01206                     currPoint = mov.Point;
01207                 }
01208                 else if (seg is LineSegment line)
01209                 {
01210                     double lengthSq = (line.Point.X - currPoint.X) * (line.Point.X - currPoint.X) +
01211                                       (line.Point.Y - currPoint.Y) * (line.Point.Y - currPoint.Y);
01212
01213                     if (lengthSq < maxLengthSq)
01214                     {
01215                         shortLinearisedPath.LineTo(line.Point);
01216                     }
01217                     else
01218                     {
01219                         int segmentCount = (int)Math.Ceiling(Math.Sqrt(lengthSq / maxLengthSq));
01220
01221                         for (int j = 0; j < segmentCount - 1; j++)
01222                         {
01223                             Point endPoint = new Point(currPoint.X + (line.Point.X - currPoint.X) * (j
      + 1) / segmentCount,
01224                                                        currPoint.Y + (line.Point.Y - currPoint.Y) * (j
      + 1) / segmentCount);
01225                             shortLinearisedPath.LineTo(endPoint);
01226                         }
01227
01228                         shortLinearisedPath.LineTo(line.Point);
01229                     }
01230                     currPoint = line.Point;
01231                 }
01232                 else if (seg is CloseSegment close)
01233                 {
01234                     double lengthSq = (startFigurePoint.X - currPoint.X) * (startFigurePoint.X -
      currPoint.X) +
01235                                       (startFigurePoint.Y - currPoint.Y) * (startFigurePoint.Y -
      currPoint.Y);
01236
01237                     if (lengthSq < maxLengthSq)
01238                     {
01239                         shortLinearisedPath.Close();
01240                     }
01241                     else
01242                     {
01243                         int segmentCount = (int)Math.Ceiling(Math.Sqrt(lengthSq / maxLengthSq));
01244
01245                         for (int j = 0; j < segmentCount - 1; j++)
01246                         {
01247                             Point endPoint = new Point(currPoint.X + (startFigurePoint.X -
      currPoint.X) * (j + 1) / segmentCount,
01248                                                        currPoint.Y + (startFigurePoint.Y -
      currPoint.Y) * (j + 1) / segmentCount);
01249                             shortLinearisedPath.LineTo(endPoint);
01250                         }
01251
```

```
01252                       shortLinearisedPath.Close();
01253                   }
01254                   currPoint = startFigurePoint;
01255               }
01256           }
01257
01258           return shortLinearisedPath;
01259       }
01260
01261 /// <summary>
01262 /// Creates a new <see cref="Graphics"/> object in which all the graphics actions have been
      transformed using an arbitrary transformation function.  Raster images are replaced by grey
      rectangles.
01263 /// </summary>
01264 /// <param name="transformationFunction">An arbitrary transformation function.</param>
01265 /// <param name="linearisationResolution">The resolution that will be used to linearise curve
      segments.</param>
01266 /// <param name="maxSegmentLength">The maximum length of line segments.</param>
01267 /// <returns>A new <see cref="Graphics"/> object in which all graphics actions have been linearised
      and transformed using the <paramref name="transformationFunction"/>.</returns>
01268       public Graphics Transform(Func<Point, Point> transformationFunction, double
      linearisationResolution, double maxSegmentLength)
01269       {
01270           Graphics destinationGraphics = new Graphics();
01271
01272           Stack<double[,]> transformMatrix = new Stack<double[,]>();
01273           double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01274
01275           for (int i = 0; i < this.Actions.Count; i++)
01276           {
01277               if (this.Actions[i] is RectangleAction)
01278               {
01279                   RectangleAction rec = this.Actions[i] as RectangleAction;
01280
01281                   GraphicsPath rectanglePath = new GraphicsPath();
01282
01283                   Point pt1 = rec.TopLeft;
01284                   Point pt2 = new Point(rec.TopLeft.X + rec.Size.Width, rec.TopLeft.Y);
01285                   Point pt3 = new Point(rec.TopLeft.X + rec.Size.Width, rec.TopLeft.Y +
      rec.Size.Height);
01286                   Point pt4 = new Point(rec.TopLeft.X, rec.TopLeft.Y + rec.Size.Height);
01287
01288                   rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01289
01290                   rectanglePath = ReduceMaximumLength(rectanglePath, maxSegmentLength).Transform(pt
      => transformationFunction(Multiply(currMatrix, pt)));
01291
01292                   if (rec.Fill != null)
01293                   {
01294                       destinationGraphics.FillPath(rectanglePath, rec.Fill, rec.Tag);
01295                   }
01296                   else if (rec.Stroke != null)
01297                   {
01298                       destinationGraphics.StrokePath(rectanglePath, rec.Stroke, rec.LineWidth,
      rec.LineCap, rec.LineJoin, rec.LineDash, rec.Tag);
01299                   }
01300               }
01301               else if (this.Actions[i] is PathAction)
01302               {
01303                   PathAction pth = this.Actions[i] as PathAction;
01304
01305                   GraphicsPath newPath =
      ReduceMaximumLength(pth.Path.Linearise(linearisationResolution), maxSegmentLength).Transform(pt =>
      transformationFunction(Multiply(currMatrix, pt)));
01306
01307                   if (pth.IsClipping)
01308                   {
01309                       destinationGraphics.SetClippingPath(newPath);
01310                   }
01311                   else
01312                   {
01313                       if (pth.Fill != null)
01314                       {
01315                           destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01316                       }
01317                       else if (pth.Stroke != null)
01318                       {
01319                           destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
      pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01320                       }
01321                   }
01322               }
01323               else if (this.Actions[i] is TextAction)
01324               {
01325                   TextAction txt = this.Actions[i] as TextAction;
01326
01327                   GraphicsPath textPath = ReduceMaximumLength(new GraphicsPath().AddText(txt.Origin,
```

```
      txt.Text, txt.Font, txt.TextBaseline).Linearise(linearisationResolution),
      maxSegmentLength).Transform(pt => transformationFunction(Multiply(currMatrix, pt)));
01328
01329                     if (txt.Fill != null)
01330                     {
01331                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01332                     }
01333                     else if (txt.Stroke != null)
01334                     {
01335                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
      txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01336                     }
01337                 }
01338                 else if (this.Actions[i] is TransformAction)
01339                 {
01340                     TransformAction trf = this.Actions[i] as TransformAction;
01341
01342                     if (trf.Delta != null)
01343                     {
01344                         currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01345                     }
01346                     else if (trf.Angle != null)
01347                     {
01348                         currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01349                     }
01350                     else if (trf.Scale != null)
01351                     {
01352                         currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01353                     }
01354                     else if (trf.Matrix != null)
01355                     {
01356                         currMatrix = Multiply(currMatrix, trf.Matrix);
01357                     }
01358                 }
01359                 else if (this.Actions[i] is StateAction)
01360                 {
01361                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01362                     {
01363                         transformMatrix.Push(currMatrix);
01364                     }
01365                     else
01366                     {
01367                         currMatrix = transformMatrix.Pop();
01368                     }
01369                 }
01370                 else if (this.Actions[i] is RasterImageAction)
01371                 {
01372                     RasterImageAction img = this.Actions[i] as RasterImageAction;
01373
01374                     GraphicsPath rectanglePath = new GraphicsPath();
01375
01376                     Point pt1 = new Point(img.DestinationX, img.DestinationY);
01377                     Point pt2 = new Point(img.DestinationX + img.DestinationWidth, img.DestinationY);
01378                     Point pt3 = new Point(img.DestinationX + img.DestinationWidth, img.DestinationY +
      img.DestinationHeight);
01379                     Point pt4 = new Point(img.DestinationX, img.DestinationY + img.DestinationHeight);
01380
01381                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01382
01383                     rectanglePath = ReduceMaximumLength(rectanglePath, maxSegmentLength).Transform(pt
      => transformationFunction(Multiply(currMatrix, pt)));
01384
01385                     destinationGraphics.FillPath(rectanglePath, Colour.FromRgb(220, 220, 220),
      img.Tag);
01386                 }
01387             }
01388
01389             return destinationGraphics;
01390         }
01391
01392 /// <summary>
01393 /// Creates a new <see cref="Graphics"/> object by linearising all of the elements of the current
      instance, i.e.  replacing curve segments with series of line segments that approximate them.  Raster
      images are left unchanged.
01394 /// </summary>
01395 /// <param name="resolution">The resolution that will be used to linearise curve segments.</param>
01396 /// <returns>A new <see cref="Graphics"/> object containing the linearised elements.</returns>
01397         public Graphics Linearise(double resolution)
01398         {
01399             Graphics destinationGraphics = new Graphics();
01400
01401             for (int i = 0; i < this.Actions.Count; i++)
01402             {
01403                 if (this.Actions[i] is RectangleAction || this.Actions[i] is TransformAction ||
```

```
this.Actions[i] is StateAction || this.Actions[i] is RasterImageAction)
01404                 {
01405                     destinationGraphics.Actions.Add(this.Actions[i]);
01406                 }
01407                 else if (this.Actions[i] is PathAction)
01408                 {
01409                     PathAction pth = this.Actions[i] as PathAction;
01410 
01411                     GraphicsPath newPath = pth.Path.Linearise(resolution);
01412 
01413                     if (pth.IsClipping)
01414                     {
01415                         destinationGraphics.SetClippingPath(newPath);
01416                     }
01417                     else
01418                     {
01419                         if (pth.Fill != null)
01420                         {
01421                             destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01422                         }
01423                         else if (pth.Stroke != null)
01424                         {
01425                             destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01426                         }
01427                     }
01428                 }
01429                 else if (this.Actions[i] is TextAction)
01430                 {
01431                     TextAction txt = this.Actions[i] as TextAction;
01432 
01433                     GraphicsPath textPath = new GraphicsPath().AddText(txt.Origin, txt.Text, txt.Font,
txt.TextBaseline).Linearise(resolution);
01434 
01435                     if (txt.Fill != null)
01436                     {
01437                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01438                     }
01439                     else if (txt.Stroke != null)
01440                     {
01441                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01442                     }
01443                 }
01444             }
01445 
01446             return destinationGraphics;
01447         }
01448 
01449 /// <summary>
01450 /// Computes the rectangular bounds of the region affected by the drawing operations performed on the
<see cref="Graphics"/> object.
01451 /// </summary>
01452 /// <returns>The smallest rectangle that contains all the elements drawn on the <see
cref="Graphics"/>.</returns>
01453         public Rectangle GetBounds()
01454         {
01455             Rectangle tbr = Rectangle.NaN;
01456             bool initialised = false;
01457 
01458             Stack<Rectangle> clippingPaths = new Stack<Rectangle>();
01459             Rectangle currClippingPath = Rectangle.NaN;
01460 
01461             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01462             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01463 
01464             for (int i = 0; i < this.Actions.Count; i++)
01465             {
01466                 if (this.Actions[i] is IPrintableAction act)
01467                 {
01468                     Rectangle bounds = act.GetBounds();
01469 
01470                     double lineThickness = double.IsNaN(act.LineWidth) ?  0 :  act.LineWidth;
01471 
01472                     Point pt1 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
0.5, bounds.Location.Y - lineThickness * 0.5));
01473                     Point pt2 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
lineThickness * 0.5, bounds.Location.Y - lineThickness * 0.5));
01474                     Point pt3 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
lineThickness * 0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01475                     Point pt4 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01476 
01477                     bounds = Point.Bounds(pt1, pt2, pt3, pt4);
01478 
01479                     if (act is PathAction pth && pth.IsClipping)
01480                     {
```

```
01481                           currClippingPath = bounds;
01482                       }
01483                       else
01484                       {
01485                           if (!currClippingPath.IsNaN())
01486                           {
01487                               bounds = Rectangle.Intersection(bounds, currClippingPath);
01488                           }
01489
01490                           if (!double.IsNaN(bounds.Location.X) && !double.IsNaN(bounds.Location.Y) &&
      !double.IsNaN(bounds.Size.Width) && !double.IsNaN(bounds.Size.Height))
01491                           {
01492                               if (!initialised)
01493                               {
01494                                   tbr = bounds;
01495                                   initialised = true;
01496                               }
01497                               else
01498                               {
01499                                   tbr = Rectangle.Union(tbr, bounds);
01500                               }
01501                           }
01502                       }
01503                   }
01504                   else if (this.Actions[i] is TransformAction)
01505                   {
01506                       TransformAction trf = this.Actions[i] as TransformAction;
01507
01508                       if (trf.Delta != null)
01509                       {
01510                           currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01511                       }
01512                       else if (trf.Angle != null)
01513                       {
01514                           currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01515                       }
01516                       else if (trf.Scale != null)
01517                       {
01518                           currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01519                       }
01520                       else if (trf.Matrix != null)
01521                       {
01522                           currMatrix = Multiply(currMatrix, trf.Matrix);
01523                       }
01524                   }
01525                   else if (this.Actions[i] is StateAction)
01526                   {
01527                       if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01528                       {
01529                           transformMatrix.Push(currMatrix);
01530                           clippingPaths.Push(currClippingPath);
01531                       }
01532                       else
01533                       {
01534                           currMatrix = transformMatrix.Pop();
01535                           currClippingPath = clippingPaths.Pop();
01536                       }
01537                   }
01538               }
01539
01540               return tbr;
01541           }
01542
01543 /// <summary>
01544 /// A method that is used to rasterise a region of a <see cref="Graphics"/> object.  Set this to <see
      langword="null"/> if you wish to use the default
01545 /// rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp).
      You will have to provide your own implementation
01546 /// of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your
      project.  The first argument of this method is
01547 /// the <see cref="Graphics"/> to be rasterised, the second is a <see cref="Rectangle"/> representing
      the region to rasterise, the third is a
01548 /// <see cref="double"/> representing the scale, and the third is a boolean value indicating whether
      the resulting <see cref="RasterImage"/> should
01549 /// be interpolated.
01550 /// </summary>
01551           public static Func<Graphics, Rectangle, double, bool, RasterImage> RasterisationMethod = null;
01552
01553 /// <summary>
01554 /// Tries to rasterise specified region of this <see cref="Graphics"/> object using the default
      rasterisation method.
01555 /// </summary>
01556 /// <param name="region">The region of the <see cref="Graphics"/> to rasterise.</param>
01557 /// <param name="scale">The scale at which the image is rasterised.</param>
```

```
01558 /// <param name="interpolate">Determines whether the resulting <see cref="RasterImage"/> should be
      interpolated or not.</param>
01559 /// <param name="output">When this method returns, this will contain the rasterised image (or <see
      langword="null"/> if the image could not be rasterised.</param>
01560 /// <returns><see langword="true"/> if the image could be rasterised; <see langword="false"/> if it
      could not be rasterised.</returns>
01561         public bool TryRasterise(Rectangle region, double scale, bool interpolate, out RasterImage
      output)
01562         {
01563             if (RasterisationMethod != null)
01564             {
01565                 output = RasterisationMethod(this, region, scale, interpolate);
01566                 return true;
01567             }
01568             else
01569             {
01570                 System.Reflection.Assembly raster;
01571 
01572                 try
01573                 {
01574                     raster = System.Reflection.Assembly.Load("VectSharp.Raster");
01575                     if (raster != null)
01576                     {
01577                         System.Reflection.MethodInfo rasteriser =
      raster.GetType("VectSharp.Raster.Raster").GetMethod("Rasterise", System.Reflection.BindingFlags.Public
      | System.Reflection.BindingFlags.Static);
01578                         output = (RasterImage)rasteriser.Invoke(null, new object[] { this, region,
      scale, interpolate });
01579                         return true;
01580                     }
01581                 }
01582                 catch { }
01583 
01584                 try
01585                 {
01586                     raster = System.Reflection.Assembly.Load("VectSharp.Raster.ImageSharp");
01587                     if (raster != null)
01588                     {
01589                         System.Reflection.MethodInfo rasteriser =
      raster.GetType("VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter").GetMethod("Rasterise",
      System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Static);
01590                         output = (RasterImage)rasteriser.Invoke(null, new object[] { this, region,
      scale, interpolate });
01591                         return true;
01592                     }
01593                 }
01594                 catch { }
01595 
01596 
01597                 output = null;
01598                 return false;
01599             }
01600         }
01601 
01602 /// <summary>
01603 /// Removes graphics actions that fall completely outside of the specified <paramref name="region"/>.
01604 /// </summary>
01605 /// <param name="region">The area to preserve.</param>
01606         public void Crop(Rectangle region)
01607         {
01608             List<int> itemsToRemove = new List<int>();
01609 
01610             Stack<Rectangle> clippingPaths = new Stack<Rectangle>();
01611             Rectangle currClippingPath = Rectangle.NaN;
01612 
01613             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01614             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01615 
01616             for (int i = 0; i < this.Actions.Count; i++)
01617             {
01618                 if (this.Actions[i] is IPrintableAction act)
01619                 {
01620                     Rectangle bounds = act.GetBounds();
01621 
01622                     double lineThickness = double.IsNaN(act.LineWidth) ?  0 :  act.LineWidth;
01623 
01624                     Point pt1 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y - lineThickness * 0.5));
01625                     Point pt2 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y - lineThickness * 0.5));
01626                     Point pt3 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01627                     Point pt4 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01628 
01629                     bounds = Point.Bounds(pt1, pt2, pt3, pt4);
01630 
```

```
01631                     if (act is PathAction pth && pth.IsClipping)
01632                     {
01633                         currClippingPath = bounds;
01634                     }
01635                     else
01636                     {
01637                         if (!currClippingPath.IsNaN())
01638                         {
01639                             bounds = Rectangle.Intersection(bounds, currClippingPath);
01640                         }
01641 
01642                         if (double.IsNaN(bounds.Location.X) || double.IsNaN(bounds.Location.Y) ||
      double.IsNaN(bounds.Size.Width) || double.IsNaN(bounds.Size.Height) || Rectangle.Intersection(bounds,
      region).IsNaN())
01643                         {
01644                             itemsToRemove.Add(i);
01645                         }
01646                     }
01647                 }
01648                 else if (this.Actions[i] is TransformAction)
01649                 {
01650                     TransformAction trf = this.Actions[i] as TransformAction;
01651 
01652                     if (trf.Delta != null)
01653                     {
01654                         currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01655                     }
01656                     else if (trf.Angle != null)
01657                     {
01658                         currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01659                     }
01660                     else if (trf.Scale != null)
01661                     {
01662                         currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01663                     }
01664                     else if (trf.Matrix != null)
01665                     {
01666                         currMatrix = Multiply(currMatrix, trf.Matrix);
01667                     }
01668                 }
01669                 else if (this.Actions[i] is StateAction)
01670                 {
01671                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01672                     {
01673                         transformMatrix.Push(currMatrix);
01674                         clippingPaths.Push(currClippingPath);
01675                     }
01676                     else
01677                     {
01678                         currMatrix = transformMatrix.Pop();
01679                         currClippingPath = clippingPaths.Pop();
01680                     }
01681                 }
01682             }
01683 
01684             for (int i = itemsToRemove.Count - 1; i >= 0; i--)
01685             {
01686                 this.Actions.RemoveAt(itemsToRemove[i]);
01687             }
01688         }
01689 
01690 /// <summary>
01691 /// Removes graphics actions that fall completely outside of the specified region.
01692 /// </summary>
01693 /// <param name="topLeft">The top-left corner of the area to preserve.</param>
01694 /// <param name="size">The size of the area to preserve.</param>
01695         public void Crop(Point topLeft, Size size)
01696         {
01697             Rectangle region = new Rectangle(topLeft, size);
01698             this.Crop(region);
01699         }
01700 
01701 /// <summary>
01702 /// Gets all the tags that have been defined in the <see cref="Graphics"/>.
01703 /// </summary>
01704 /// <returns>An <see cref="IEnumerable{T}"/> of <see cref="string"/>s that, when enumerated, returns
      all the tags that have been defined in the <see cref="Graphics"/>.</returns>
01705         public IEnumerable<string> GetTags()
01706         {
01707             foreach (IGraphicsAction action in this.Actions)
01708             {
01709                 if (!string.IsNullOrEmpty(action.Tag))
01710                 {
01711                     yield return action.Tag;
```

```
01712                 }
01713             }
01714         }
01715     }
01716 }
```

8.67 Graphics.Text.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp
00022 {
00023     public partial class Graphics
00024     {
00025 /// <summary>
00026 /// Fill a text string.
00027 /// </summary>
00028 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00029 /// <param name="text">The string to draw.</param>
00030 /// <param name="font">The font with which to draw the text.</param>
00031 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00032 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00033 /// <param name="tag">A tag to identify the filled text.</param>
00034         public void FillText(Point origin, string text, Font font, Brush fillColour, TextBaselines
      textBaseline = TextBaselines.Top, string tag = null)
00035         {
00036             if (!string.IsNullOrEmpty(text))
00037             {
00038                 Actions.Add(new TextAction(origin, text, font, textBaseline, fillColour, null, 0,
      LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00039
00040                 if (font.Underline != null)
00041                 {
00042                     FillTextUnderline(origin, text, font, fillColour, textBaseline,
      (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :  tag);
00043                 }
00044             }
00045         }
00046
00047 /// <summary>
00048 /// Fill a text string.
00049 /// </summary>
00050 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00051 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00052 /// <param name="text">The string to draw.</param>
00053 /// <param name="font">The font with which to draw the text.</param>
00054 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00055 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00056 /// <param name="tag">A tag to identify the filled text.</param>
00057         public void FillText(double originX, double originY, string text, Font font, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00058         {
00059             if (!string.IsNullOrEmpty(text))
00060             {
00061                 Actions.Add(new TextAction(new Point(originX, originY), text, font, textBaseline,
      fillColour, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00062
00063                 if (font.Underline != null)
00064                 {
00065                     FillTextUnderline(originX, originY, text, font, fillColour, textBaseline,
      (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :  tag);
00066                 }
```

```
00067             }
00068         }
00069 
00070 /// <summary>
00071 /// Stroke a text string.
00072 /// </summary>
00073 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00074 /// <param name="text">The string to draw.</param>
00075 /// <param name="font">The font with which to draw the text.</param>
00076 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00077 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00078 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00079 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00080 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00081 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00082 /// <param name="tag">A tag to identify the stroked text.</param>
00083         public void StrokeText(Point origin, string text, Font font, Brush strokeColour, TextBaselines
      textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins
      lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00084         {
00085             if (!string.IsNullOrEmpty(text))
00086             {
00087                 Actions.Add(new TextAction(origin, text, font, textBaseline, null, strokeColour,
      lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00088 
00089                 if (font.Underline != null)
00090                 {
00091                     StrokeTextUnderline(origin, text, font, strokeColour, textBaseline, lineWidth,
      lineCap, lineJoin, lineDash, (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :
      tag);
00092                 }
00093             }
00094         }
00095 
00096 /// <summary>
00097 /// Stroke a text string.
00098 /// </summary>
00099 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00100 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00101 /// <param name="text">The string to draw.</param>
00102 /// <param name="font">The font with which to draw the text.</param>
00103 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00104 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00105 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00106 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00107 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00108 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00109 /// <param name="tag">A tag to identify the stroked text.</param>
00110         public void StrokeText(double originX, double originY, string text, Font font, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00111         {
00112             if (!string.IsNullOrEmpty(text))
00113             {
00114                 Actions.Add(new TextAction(new Point(originX, originY), text, font, textBaseline,
      null, strokeColour, lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00115 
00116                 if (font.Underline != null)
00117                 {
00118                     StrokeTextUnderline(originX, originY, text, font, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag +
      "@underline") :  tag);
00119                 }
00120             }
00121         }
00122 
00123 /// <summary>
00124 /// Fill a text string along a <see cref="GraphicsPath"/>.
00125 /// </summary>
00126 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00127 /// <param name="text">The string to draw.</param>
00128 /// <param name="font">The font with which to draw the text.</param>
00129 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00130 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
00131 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00132 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00133 /// <param name="tag">A tag to identify the filled text.</param>
00134         public void FillTextOnPath(GraphicsPath path, string text, Font font, Brush fillColour, double
      reference = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline = TextBaselines.Top,
      string tag = null)
00135         {
```

```
00136             if (!string.IsNullOrEmpty(text))
00137             {
00138                 if (font.Underline != null)
00139                 {
00140                     font = new Font(font.FontFamily, font.FontSize, false);
00141                 }
00142 
00143                 double currDelta = 0;
00144                 double pathLength = path.MeasureLength();
00145 
00146                 Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00147 
00148                 switch (anchor)
00149                 {
00150                     case TextAnchors.Left:
00151                         break;
00152                     case TextAnchors.Center:
00153                         currDelta = -fullMetrics.Width * 0.5 / pathLength;
00154                         break;
00155                     case TextAnchors.Right:
00156                         currDelta = -fullMetrics.Width / pathLength;
00157                         break;
00158                 }
00159 
00160                 Point currentGlyphPlacementDelta = new Point();
00161                 Point currentGlyphAdvanceDelta = new Point();
00162                 Point nextGlyphPlacementDelta = new Point();
00163                 Point nextGlyphAdvanceDelta = new Point();
00164 
00165                 for (int i = 0; i < text.Length; i++)
00166                 {
00167                     string c = text.Substring(i, 1);
00168 
00169                     if (Font.EnableKerning && i < text.Length - 1)
00170                     {
00171                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00172                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00173                         nextGlyphAdvanceDelta = new Point();
00174                         nextGlyphPlacementDelta = new Point();
00175 
00176                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00177 
00178                         if (kerning != null)
00179                         {
00180                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00181                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00182 
00183                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00184                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00185                         }
00186                     }
00187 
00188                     Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00189 
00190                     Point origin = path.GetPointAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000);
00191 
00192                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing) / pathLength * 0.5);
00193 
00194                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
      font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00195 
00196                     this.Save();
00197 
00198                     this.Translate(origin);
00199                     this.Rotate(Math.Atan2(tangent.Y, tangent.X));
00200 
00201                     string glyphTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00202 
00203                     switch (textBaseline)
00204                     {
00205                         case TextBaselines.Top:
00206                             if (i > 0)
00207                             {
00208                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Top), c,
      font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00209                             }
00210                             else
00211                             {
```

```
00212                                 this.FillText(new Point(0, fullMetrics.Top), c, font, fillColour,
      textBaseline:  TextBaselines.Baseline, glyphTag);
00213                             }
00214                             break;
00215                         case TextBaselines.Baseline:
00216                             if (i > 0)
00217                             {
00218                                 this.FillText(new Point(metrics.LeftSideBearing, 0), c, font,
      fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00219                             }
00220                             else
00221                             {
00222                                 this.FillText(new Point(0, 0), c, font, fillColour, textBaseline:
      TextBaselines.Baseline, glyphTag);
00223                             }
00224                             break;
00225                         case TextBaselines.Bottom:
00226                             if (i > 0)
00227                             {
00228                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom),
      c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00229                             }
00230                             else
00231                             {
00232                                 this.FillText(new Point(0, fullMetrics.Bottom), c, font, fillColour,
      textBaseline:  TextBaselines.Baseline, glyphTag);
00233                             }
00234                             break;
00235                         case TextBaselines.Middle:
00236                             if (i > 0)
00237                             {
00238                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom +
      fullMetrics.Height / 2), c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00239                             }
00240                             else
00241                             {
00242                                 this.FillText(new Point(0, fullMetrics.Bottom + fullMetrics.Height /
      2), c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00243                             }
00244                             break;
00245                     }
00246
00247                     this.Restore();
00248
00249                     if (i > 0)
00250                     {
00251                         currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00252                     }
00253                     else
00254                     {
00255                         currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00256                     }
00257                 }
00258             }
00259         }
00260
00261 /// <summary>
00262 /// Stroke a text string along a <see cref="GraphicsPath"/>.
00263 /// </summary>
00264 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00265 /// <param name="text">The string to draw.</param>
00266 /// <param name="font">The font with which to draw the text.</param>
00267 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00268 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00269 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00270 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00271 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00272 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
00273 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00274 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00275 /// <param name="tag">A tag to identify the stroked text.</param>
00276         public void StrokeTextOnPath(GraphicsPath path, string text, Font font, Brush strokeColour,
      double reference = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline =
      TextBaselines.Top, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin =
      LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00277         {
00278             if (!string.IsNullOrEmpty(text))
00279             {
00280                 if (font.Underline != null)
00281                 {
00282                     font = new Font(font.FontFamily, font.FontSize, false);
00283                 }
```

```
00284
00285                 double currDelta = 0;
00286                 double pathLength = path.MeasureLength();
00287
00288                 Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00289
00290                 switch (anchor)
00291                 {
00292                     case TextAnchors.Left:
00293                         break;
00294                     case TextAnchors.Center:
00295                         currDelta = -fullMetrics.Width * 0.5 / pathLength;
00296                         break;
00297                     case TextAnchors.Right:
00298                         currDelta = -fullMetrics.Width / pathLength;
00299                         break;
00300                 }
00301
00302                 Point currentGlyphPlacementDelta = new Point();
00303                 Point currentGlyphAdvanceDelta = new Point();
00304                 Point nextGlyphPlacementDelta = new Point();
00305                 Point nextGlyphAdvanceDelta = new Point();
00306
00307                 for (int i = 0; i < text.Length; i++)
00308                 {
00309                     string c = text.Substring(i, 1);
00310
00311                     if (Font.EnableKerning && i < text.Length - 1)
00312                     {
00313                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00314                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00315                         nextGlyphAdvanceDelta = new Point();
00316                         nextGlyphPlacementDelta = new Point();
00317
00318                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00319
00320                         if (kerning != null)
00321                         {
00322                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00323                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00324
00325                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00326                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00327                         }
00328                     }
00329
00330                     Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00331
00332                     Point origin = path.GetPointAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000);
00333
00334                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing) / pathLength * 0.5);
00335
00336                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
      font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00337
00338                     this.Save();
00339
00340                     this.Translate(origin);
00341                     this.Rotate(Math.Atan2(tangent.Y, tangent.X));
00342
00343                     string glyphTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00344
00345                     switch (textBaseline)
00346                     {
00347                         case TextBaselines.Top:
00348                             if (i > 0)
00349                             {
00350                                 this.StrokeText(new Point(metrics.LeftSideBearing, fullMetrics.Top),
      c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00351                             }
00352                             else
00353                             {
00354                                 this.StrokeText(new Point(0, fullMetrics.Top), c, font, strokeColour,
      textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, glyphTag);
00355                             }
00356                             break;
00357                         case TextBaselines.Baseline:
```

```
00358                               if (i > 0)
00359                               {
00360                                   this.StrokeText(new Point(metrics.LeftSideBearing, 0), c, font,
      strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00361                               }
00362                               else
00363                               {
00364                                   this.StrokeText(new Point(0, 0), c, font, strokeColour, textBaseline:
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, glyphTag);
00365                               }
00366                               break;
00367                           case TextBaselines.Bottom:
00368                               if (i > 0)
00369                               {
00370                                   this.StrokeText(new Point(metrics.LeftSideBearing,
      fullMetrics.Bottom), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap,
      lineJoin, lineDash, glyphTag);
00371                               }
00372                               else
00373                               {
00374                                   this.StrokeText(new Point(0, fullMetrics.Bottom), c, font,
      strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00375                               }
00376                               break;
00377                           case TextBaselines.Middle:
00378                               if (i > 0)
00379                               {
00380                                   this.StrokeText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom
      + fullMetrics.Height / 2), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, glyphTag);
00381                               }
00382                               else
00383                               {
00384                                   this.StrokeText(new Point(0, fullMetrics.Bottom + fullMetrics.Height /
      2), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin,
      lineDash, glyphTag);
00385                               }
00386                               break;
00387                       }
00388
00389                       this.Restore();
00390
00391                       if (i > 0)
00392                       {
00393                           currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00394                       }
00395                       else
00396                       {
00397                           currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00398                       }
00399                   }
00400               }
00401           }
00402
00403 /// <summary>
00404 /// Fill a formatted text string.
00405 /// </summary>
00406 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00407 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00408 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the text.  This can be
      overridden by each <paramref name="text"/> element.</param>
00409 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00410 /// <param name="tag">A tag to identify the filled text.</param>
00411         public void FillText(Point origin, IEnumerable<FormattedText> text, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00412         {
00413             if (text != null)
00414             {
00415                 List<FormattedText> enumeratedText = new List<FormattedText>();
00416                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00417
00418                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00419
00420                 if (enumeratedText.Count > 0)
00421                 {
00422                     Point baselineOrigin = origin;
00423
00424                     switch (textBaseline)
00425                     {
00426                         case TextBaselines.Baseline:
00427                             baselineOrigin = origin;
00428                             break;
```

```
00429                           case TextBaselines.Top:
00430                               baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00431                               break;
00432                           case TextBaselines.Bottom:
00433                               baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00434                               break;
00435                           case TextBaselines.Middle:
00436                               baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00437                               break;
00438                       }
00439
00440                       for (int i = 0; i < enumeratedText.Count; i++)
00441                       {
00442                           FormattedText txt = enumeratedText[i];
00443
00444                           string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00445
00446                           if (txt.Script == Script.Normal)
00447                           {
00448                               Font.DetailedFontMetrics metrics = allMetrics[i];
00449
00450                               if (i > 0)
00451                               {
00452                                   FillText(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y,
      txt.Text, txt.Font, txt.Brush ??  fillColour, TextBaselines.Baseline, spanTag);
00453                               }
00454                               else
00455                               {
00456                                   FillText(baselineOrigin, txt.Text, txt.Font, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00457                               }
00458
00459                               if (i > 0)
00460                               {
00461                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00462                               }
00463                               else
00464                               {
00465                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00466                               }
00467                           }
00468                           else
00469                           {
00470                               Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00471
00472                               Font.DetailedFontMetrics metrics = allMetrics[i];
00473
00474                               if (i == 0)
00475                               {
00476                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00477                               }
00478
00479                               if (txt.Script == Script.Subscript)
00480                               {
00481                                   FillText(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y
      + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  fillColour, TextBaselines.Baseline,
      spanTag);
00482                               }
00483                               else if (txt.Script == Script.Superscript)
00484                               {
00485                                   FillText(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y
      - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  fillColour, TextBaselines.Baseline,
      spanTag);
00486                               }
00487
00488
00489                               if (i > 0)
00490                               {
00491                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00492                               }
00493                               else
00494                               {
00495                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00496                               }
00497                           }
00498                       }
00499                   }
00500               }
00501           }
```

```
00502
00503 /// <summary>
00504 /// Fill a formatted text string.
00505 /// </summary>
00506 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00507 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00508 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00509 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the text.  This can be
      overridden by each <paramref name="text"/> element.</param>
00510 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00511 /// <param name="tag">A tag to identify the filled text.</param>
00512         public void FillText(double originX, double originY, IEnumerable<FormattedText> text, Brush
      fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00513         {
00514             FillText(new Point(originX, originY), text, fillColour, textBaseline, tag);
00515         }
00516
00517 /// <summary>
00518 /// Stroke a formatted text string.
00519 /// </summary>
00520 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00521 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00522 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the text.</param>
00523 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00524 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00525 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00526 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00527 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00528 /// <param name="tag">A tag to identify the stroked text.</param>
00529         public void StrokeText(Point origin, IEnumerable<FormattedText> text, Brush strokeColour,
      TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00530         {
00531             if (text != null)
00532             {
00533                 List<FormattedText> enumeratedText = new List<FormattedText>();
00534                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00535
00536                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00537                 if (enumeratedText.Count > 0)
00538                 {
00539                     Point baselineOrigin = origin;
00540
00541                     switch (textBaseline)
00542                     {
00543                         case TextBaselines.Baseline:
00544                             baselineOrigin = origin;
00545                             break;
00546                         case TextBaselines.Top:
00547                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00548                             break;
00549                         case TextBaselines.Bottom:
00550                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00551                             break;
00552                         case TextBaselines.Middle:
00553                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00554                             break;
00555                     }
00556
00557                     for (int i = 0; i < enumeratedText.Count; i++)
00558                     {
00559                         FormattedText txt = enumeratedText[i];
00560
00561                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00562
00563                         if (txt.Script == Script.Normal)
00564                         {
00565                             Font.DetailedFontMetrics metrics = allMetrics[i];
00566
00567                             if (i > 0)
00568                             {
00569                                 StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  strokeColour, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, spanTag);
00570                             }
00571                             else
00572                             {
00573                                 StrokeText(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      strokeColour, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00574                             }
00575
00576                             if (i > 0)
```

```
00577                               {
00578                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00579                               }
00580                               else
00581                               {
00582                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00583                               }
00584                           }
00585                           else
00586                           {
00587                               Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00588
00589                               Font.DetailedFontMetrics metrics = allMetrics[i];
00590
00591                               if (i == 0)
00592                               {
00593                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00594                               }
00595
00596                               if (txt.Script == Script.Subscript)
00597                               {
00598                                   StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00599                               }
00600                               else if (txt.Script == Script.Superscript)
00601                               {
00602                                   StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00603                               }
00604
00605
00606                               if (i > 0)
00607                               {
00608                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00609                               }
00610                               else
00611                               {
00612                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00613                               }
00614                           }
00615                       }
00616                   }
00617               }
00618           }
00619
00620 /// <summary>
00621 /// Stroke a formatted text string.
00622 /// </summary>
00623 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00624 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00625 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00626 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the text.</param>
00627 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00628 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00629 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00630 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00631 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00632 /// <param name="tag">A tag to identify the stroked text.</param>
00633         public void StrokeText(double originX, double originY, IEnumerable<FormattedText> text, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00634         {
00635             StrokeText(new Point(originX, originY), text, strokeColour, textBaseline, lineWidth,
      lineCap, lineJoin, lineDash, tag);
00636         }
00637
00638 /// <summary>
00639 /// Measure a text string.
00640 /// See also <seealso cref="Font.MeasureText(string)"/> and <seealso
      cref="Font.MeasureTextAdvanced(string)"/>.
00641 /// </summary>
00642 /// <param name="text">The string to measure.</param>
00643 /// <param name="font">The font to use to measure the string.</param>
00644 /// <returns>The size of the measured <paramref name="text"/>.</returns>
00645         public Size MeasureText(string text, Font font)
00646         {
00647             if (!string.IsNullOrEmpty(text))
```

```
00648             {
00649                 return font.MeasureText(text);
00650             }
00651             else
00652             {
00653                 return new Size(0, 0);
00654             }
00655         }
00656 
00657 /// <summary>
00658 /// Measure a formatted text string.
00659 /// See also <seealso cref="FormattedTextExtensions.Measure(IEnumerable{FormattedText})"/>.
00660 /// </summary>
00661 /// <param name="text">The collection of <see cref="FormattedText"/> objects to measure.</param>
00662 /// <returns>The size of the measured <paramref name="text"/>.</returns>
00663         public Size MeasureText(IEnumerable<FormattedText> text)
00664         {
00665             Font.DetailedFontMetrics metrics = text.Measure();
00666 
00667             return new Size(metrics.Width, metrics.Height);
00668         }
00669 
00670 /// <summary>
00671 /// Fills the underline of the specified text string.
00672 /// </summary>
00673 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00674 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00675 /// <param name="text">The string whose underline will be draw.</param>
00676 /// <param name="font">The font with which to draw the text.</param>
00677 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the underline.</param>
00678 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00679 /// <param name="tag">A tag to identify the filled underline.</param>
00680         public void FillTextUnderline(double originX, double originY, string text, Font font, Brush
      fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00681         {
00682             FillTextUnderline(new Point(originX, originY), text, font, fillColour, textBaseline, tag);
00683         }
00684 
00685 /// <summary>
00686 /// Fills the underline of the specified text string.
00687 /// </summary>
00688 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00689 /// <param name="text">The string whose underline will be draw.</param>
00690 /// <param name="font">The font with which to draw the text.</param>
00691 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the underline.</param>
00692 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00693 /// <param name="tag">A tag to identify the filled underline.</param>
00694         public void FillTextUnderline(Point origin, string text, Font font, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00695         {
00696             if (!string.IsNullOrEmpty(text))
00697             {
00698                 GraphicsPath underline = new GraphicsPath().AddTextUnderline(origin, text, font,
      textBaseline);
00699                 FillPath(underline, fillColour, tag);
00700             }
00701         }
00702 
00703 /// <summary>
00704 /// Stroke the underline of the specified text string.
00705 /// </summary>
00706 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00707 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00708 /// <param name="text">The string whose underline will be drawn.</param>
00709 /// <param name="font">The font with which to draw the text.</param>
00710 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the underline.</param>
00711 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00712 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00713 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
00714 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00715 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00716 /// <param name="tag">A tag to identify the stroked underline.</param>
00717         public void StrokeTextUnderline(double originX, double originY, string text, Font font, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00718         {
00719             StrokeTextUnderline(new Point(originX, originY), text, font, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, tag);
00720         }
00721 
00722 /// <summary>
00723 /// Stroke the underline of the specified text string.
```

```
00724 /// </summary>
00725 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00726 /// <param name="text">The string whose underline will be drawn.</param>
00727 /// <param name="font">The font with which to draw the text.</param>
00728 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the underline.</param>
00729 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00730 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00731 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
00732 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00733 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00734 /// <param name="tag">A tag to identify the stroked underline.</param>
00735         public void StrokeTextUnderline(Point origin, string text, Font font, Brush strokeColour,
      TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00736         {
00737             if (!string.IsNullOrEmpty(text))
00738             {
00739                 GraphicsPath underline = new GraphicsPath().AddTextUnderline(origin, text, font,
      textBaseline);
00740                 StrokePath(underline, strokeColour, lineWidth, lineCap, lineJoin, lineDash, tag);
00741             }
00742         }
00743
00744
00745 /// <summary>
00746 /// Fill the underline of the specified formatted text string.
00747 /// </summary>
00748 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00749 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00750 /// <param name="text">The <see cref="FormattedText"/> whose underline will be drawn.</param>
00751 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the underline.  This can
      be overridden by each <paramref name="text"/> element.</param>
00752 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00753 /// <param name="tag">A tag to identify the filled underlined.</param>
00754         public void FillTextUnderline(double originX, double originY, IEnumerable<FormattedText> text,
      Brush fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00755         {
00756             FillTextUnderline(new Point(originX, originY), text, fillColour, textBaseline, tag);
00757         }
00758
00759 /// <summary>
00760 /// Fill the underline of the specified formatted text string.
00761 /// </summary>
00762 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00763 /// <param name="text">The <see cref="FormattedText"/> whose underline will be drawn.</param>
00764 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the underline.  This can
      be overridden by each <paramref name="text"/> element.</param>
00765 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00766 /// <param name="tag">A tag to identify the filled underlined.</param>
00767         public void FillTextUnderline(Point origin, IEnumerable<FormattedText> text, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00768         {
00769             if (text != null)
00770             {
00771
00772                 List<FormattedText> enumeratedText = new List<FormattedText>();
00773                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00774
00775                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00776
00777                 if (enumeratedText.Count > 0)
00778                 {
00779
00780                     Point baselineOrigin = origin;
00781
00782                     switch (textBaseline)
00783                     {
00784                         case TextBaselines.Baseline:
00785                             baselineOrigin = origin;
00786                             break;
00787                         case TextBaselines.Top:
00788                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00789                             break;
00790                         case TextBaselines.Bottom:
00791                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00792                             break;
00793                         case TextBaselines.Middle:
00794                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00795                             break;
00796                     }
00797
00798                     for (int i = 0; i < enumeratedText.Count; i++)
```

```
00799                     {
00800                         FormattedText txt = enumeratedText[i];
00801 
00802                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00803 
00804                         if (txt.Script == Script.Normal)
00805                         {
00806                             Font.DetailedFontMetrics metrics = allMetrics[i];
00807 
00808                             if (i > 0)
00809                             {
00810                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  fillColour, TextBaselines.Baseline, spanTag);
00811                             }
00812                             else
00813                             {
00814                                 FillTextUnderline(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      fillColour, TextBaselines.Baseline, spanTag);
00815                             }
00816 
00817                             if (i > 0)
00818                             {
00819                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00820                             }
00821                             else
00822                             {
00823                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00824                             }
00825                         }
00826                         else
00827                         {
00828                             Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00829 
00830                             Font.DetailedFontMetrics metrics = allMetrics[i];
00831 
00832                             if (i == 0)
00833                             {
00834                                 baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00835                             }
00836 
00837                             if (txt.Script == Script.Subscript)
00838                             {
00839                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00840                             }
00841                             else if (txt.Script == Script.Superscript)
00842                             {
00843                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00844                             }
00845 
00846 
00847                             if (i > 0)
00848                             {
00849                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00850                             }
00851                             else
00852                             {
00853                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00854                             }
00855                         }
00856                     }
00857                 }
00858             }
00859         }
00860 
00861 /// <summary>
00862 /// Stroke the underline of the specified formatted text string.
00863 /// </summary>
00864 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00865 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00866 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00867 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the
      underline.</param>
00868 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00869 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00870 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
```

```
00871 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00872 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00873 /// <param name="tag">A tag to identify the stroked underline.</param>
00874         public void StrokeTextUnderline(double originX, double originY, IEnumerable<FormattedText>
      text, Brush strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1,
      LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null,
      string tag = null)
00875         {
00876             StrokeTextUnderline(new Point(originX, originY), text, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, tag);
00877         }
00878
00879 /// <summary>
00880 /// Stroke the underline of the specified formatted text string.
00881 /// </summary>
00882 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00883 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00884 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the
      underline.</param>
00885 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00886 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00887 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
00888 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00889 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00890 /// <param name="tag">A tag to identify the stroked underline.</param>
00891         public void StrokeTextUnderline(Point origin, IEnumerable<FormattedText> text, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00892         {
00893             if (text != null)
00894             {
00895                 List<FormattedText> enumeratedText = new List<FormattedText>();
00896                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00897
00898                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00899
00900                 if (enumeratedText.Count > 0)
00901                 {
00902
00903                     Point baselineOrigin = origin;
00904
00905                     switch (textBaseline)
00906                     {
00907                         case TextBaselines.Baseline:
00908                             baselineOrigin = origin;
00909                             break;
00910                         case TextBaselines.Top:
00911                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00912                             break;
00913                         case TextBaselines.Bottom:
00914                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00915                             break;
00916                         case TextBaselines.Middle:
00917                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00918                             break;
00919                     }
00920
00921                     for (int i = 0; i < enumeratedText.Count; i++)
00922                     {
00923                         FormattedText txt = enumeratedText[i];
00924
00925                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00926
00927                         if (txt.Script == Script.Normal)
00928                         {
00929                             Font.DetailedFontMetrics metrics = allMetrics[i];
00930
00931                             if (i > 0)
00932                             {
00933                                 StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  strokeColour, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, spanTag);
00934                             }
00935                             else
00936                             {
00937                                 StrokeTextUnderline(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      strokeColour, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00938                             }
00939
00940                             if (i > 0)
00941                             {
00942                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
```

```
00943                               }
00944                               else
00945                               {
00946                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00947                               }
00948                           }
00949                           else
00950                           {
00951                               Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00952
00953                               Font.DetailedFontMetrics metrics = allMetrics[i];
00954
00955                               if (i == 0)
00956                               {
00957                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00958                               }
00959
00960                               if (txt.Script == Script.Subscript)
00961                               {
00962                                   StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00963                               }
00964                               else if (txt.Script == Script.Superscript)
00965                               {
00966                                   StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00967                               }
00968
00969
00970                               if (i > 0)
00971                               {
00972                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00973                               }
00974                               else
00975                               {
00976                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00977                               }
00978                           }
00979                       }
00980                   }
00981               }
00982           }
00983       }
00984   }
```

8.68 GraphicsAction.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using VectSharp.Filters;
00019
00020 namespace VectSharp
00021 {
00022     internal interface IGraphicsAction
00023     {
00024         IGraphicsAction ShallowClone();
00025         string Tag { get; set; }
00026     }
00027
00028     internal interface IPrintableAction
00029     {
```

```
00030         Brush Fill { get; }
00031         Brush Stroke { get; }
00032         double LineWidth { get; }
00033         LineCaps LineCap { get; }
00034         LineJoins LineJoin { get; }
00035         LineDash LineDash { get; }
00036         string Tag { get; set; }
00037         Rectangle GetBounds();
00038     }
00039
00040     internal class TransformAction :  IGraphicsAction
00041     {
00042         public Point?  Delta { get; } = null;
00043
00044         public double?  Angle { get; } = null;
00045
00046         public Size?  Scale { get; } = null;
00047
00048         public double[,] Matrix { get; } = null;
00049         public string Tag { get; set; }
00050
00051         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00052
00053         public TransformAction(Point delta, string tag)
00054         {
00055             this.Delta = delta;
00056             this.Tag = tag;
00057         }
00058
00059         public TransformAction(double angle, string tag)
00060         {
00061             this.Angle = angle;
00062             this.Tag = tag;
00063         }
00064
00065         public TransformAction(Size scale, string tag)
00066         {
00067             this.Scale = scale;
00068             this.Tag = tag;
00069         }
00070
00071         public TransformAction(double[,] matrix, string tag)
00072         {
00073             this.Matrix = matrix;
00074             this.Tag = tag;
00075         }
00076
00077         public double[,] GetMatrix()
00078         {
00079             if (this.Matrix != null)
00080             {
00081                 return this.Matrix;
00082             }
00083             else if (this.Delta != null)
00084             {
00085                 return Graphics.TranslationMatrix(this.Delta.Value.X, this.Delta.Value.Y);
00086             }
00087             else if (this.Angle != null)
00088             {
00089                 return Graphics.RotationMatrix(this.Angle.Value);
00090             }
00091             else if (this.Scale != null)
00092             {
00093                 return Graphics.ScaleMatrix(this.Scale.Value.Width, this.Scale.Value.Height);
00094             }
00095             else
00096             {
00097                 return null;
00098             }
00099         }
00100     }
00101
00102     internal class StateAction :  IGraphicsAction
00103     {
00104         public string Tag { get; set; } = null;
00105
00106         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00107
00108         public enum StateActionTypes
00109         {
00110             Save, Restore
00111         }
00112
00113         public StateActionTypes StateActionType { get; }
00114
00115         public StateAction(StateActionTypes type)
00116         {
```

```
00117             this.StateActionType = type;
00118         }
00119     }
00120
00121     internal class TextAction :  IGraphicsAction, IPrintableAction
00122     {
00123         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00124
00125         public Brush Fill { get; }
00126         public Brush Stroke { get; }
00127         public double LineWidth { get; }
00128         public LineCaps LineCap { get; }
00129         public LineJoins LineJoin { get; }
00130         public LineDash LineDash { get; }
00131         public string Tag { get; set; }
00132         public string Text { get; }
00133         public Point Origin { get; }
00134         public TextBaselines TextBaseline { get; }
00135         public Font Font { get; }
00136
00137         public TextAction(Point origin, string text, Font font, TextBaselines textBaseLine, Brush
      fill, Brush stroke, double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string
      tag)
00138         {
00139             this.Origin = origin;
00140             this.Text = text;
00141             this.Font = font;
00142             this.TextBaseline = textBaseLine;
00143             this.Fill = fill;
00144             this.Stroke = stroke;
00145             this.LineCap = lineCap;
00146             this.LineJoin = lineJoin;
00147             this.LineWidth = lineWidth;
00148             this.Tag = tag;
00149             this.LineDash = lineDash;
00150         }
00151
00152         public Rectangle GetBounds()
00153         {
00154             Font.DetailedFontMetrics metrics = this.Font.MeasureTextAdvanced(this.Text);
00155
00156             switch (this.TextBaseline)
00157             {
00158                 case TextBaselines.Top:
00159                     return new Rectangle(this.Origin, new Size(metrics.Width, metrics.Height));
00160                 case TextBaselines.Bottom:
00161                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Height, metrics.Width,
      metrics.Height);
00162                 case TextBaselines.Middle:
00163                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Height * 0.5,
      metrics.Width, metrics.Height);
00164                 case TextBaselines.Baseline:
00165                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Top, metrics.Width,
      metrics.Height);
00166                 default:
00167                     throw new System.ArgumentOutOfRangeException(nameof(TextBaseline),
      this.TextBaseline, "Invalid text baseline!");
00168             }
00169         }
00170     }
00171
00172     internal class RectangleAction :  IGraphicsAction, IPrintableAction
00173     {
00174         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00175         public Brush Fill { get; }
00176         public Brush Stroke { get; }
00177         public double LineWidth { get; }
00178         public LineCaps LineCap { get; }
00179         public LineJoins LineJoin { get; }
00180         public LineDash LineDash { get; }
00181         public string Tag { get; set; }
00182         public Point TopLeft { get; }
00183         public Size Size { get; }
00184
00185         public RectangleAction(Point topLeft, Size size, Brush fill, Brush stroke, double lineWidth,
      LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string tag)
00186         {
00187             this.TopLeft = topLeft;
00188             this.Size = size;
00189             this.Fill = fill;
00190             this.Stroke = stroke;
00191             this.LineCap = lineCap;
00192             this.LineJoin = lineJoin;
00193             this.LineWidth = lineWidth;
00194             this.LineDash = lineDash;
00195             this.Tag = tag;
00196         }
```

```
00197
00198         public Rectangle GetBounds()
00199         {
00200             return new Rectangle(this.TopLeft, this.Size);
00201         }
00202     }
00203
00204     internal class PathAction :  IGraphicsAction, IPrintableAction
00205     {
00206         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00207         public GraphicsPath Path { get; }
00208         public Brush Fill { get; }
00209         public Brush Stroke { get; }
00210         public string Tag { get; set; }
00211         public double LineWidth { get; }
00212         public LineCaps LineCap { get; }
00213         public LineJoins LineJoin { get; }
00214         public LineDash LineDash { get; }
00215         public bool IsClipping { get; }
00216         public FillRule FillRule { get; }
00217         public Rectangle GetBounds()
00218         {
00219             return this.Path.GetBounds();
00220         }
00221         public PathAction(GraphicsPath path, Brush fill, Brush stroke, double lineWidth, LineCaps
      lineCap, LineJoins lineJoin, LineDash lineDash, string tag, FillRule fillRule, bool isClipping)
00222         {
00223             this.Path = path;
00224             this.Fill = fill;
00225             this.Stroke = stroke;
00226             this.LineCap = lineCap;
00227             this.LineJoin = lineJoin;
00228             this.LineWidth = lineWidth;
00229             this.LineDash = lineDash;
00230             this.Tag = tag;
00231             this.IsClipping = isClipping;
00232             this.FillRule = fillRule;
00233         }
00234     }
00235
00236     internal class RasterImageAction :  IGraphicsAction, IPrintableAction
00237     {
00238         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00239         public Brush Fill { get; }
00240         public Brush Stroke { get; }
00241         public string Tag { get; set; }
00242         public double LineWidth { get; }
00243         public LineCaps LineCap { get; }
00244         public LineJoins LineJoin { get; }
00245         public LineDash LineDash { get; }
00246         public int SourceX { get; }
00247         public int SourceY { get; }
00248         public int SourceWidth { get; }
00249         public int SourceHeight { get; }
00250         public double DestinationX { get; }
00251         public double DestinationY { get; }
00252         public double DestinationWidth { get; }
00253         public double DestinationHeight { get; }
00254         public RasterImage Image { get; }
00255
00256         public RasterImageAction(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage
      image, string tag)
00257         {
00258             this.SourceX = sourceX;
00259             this.SourceY = sourceY;
00260             this.SourceWidth = sourceWidth;
00261             this.SourceHeight = sourceHeight;
00262
00263             this.DestinationX = destinationX;
00264             this.DestinationY = destinationY;
00265             this.DestinationWidth = destinationWidth;
00266             this.DestinationHeight = destinationHeight;
00267
00268             this.Image = image;
00269             this.Tag = tag;
00270         }
00271
00272         public Rectangle GetBounds()
00273         {
00274             return new Rectangle(this.DestinationX, this.DestinationY, this.DestinationWidth,
      this.DestinationHeight);
00275         }
00276     }
00277
00278     internal class FilteredGraphicsAction :  IGraphicsAction, IPrintableAction
00279     {
```

```
00280         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00281         public Brush Fill { get; }
00282         public Brush Stroke { get; }
00283         public string Tag { get; set; }
00284         public double LineWidth { get; }
00285         public LineCaps LineCap { get; }
00286         public LineJoins LineJoin { get; }
00287         public LineDash LineDash { get; }
00288         public int SourceX { get; }
00289         public int SourceY { get; }
00290         public int SourceWidth { get; }
00291         public int SourceHeight { get; }
00292         public double DestinationX { get; }
00293         public double DestinationY { get; }
00294         public double DestinationWidth { get; }
00295         public double DestinationHeight { get; }
00296         public RasterImage Image { get; }
00297         public Graphics Content { get; }
00298         public IFilter Filter { get; }
00299 
00300         public FilteredGraphicsAction(Graphics graphics, IFilter filter)
00301         {
00302             this.Content = graphics;
00303             this.Filter = filter;
00304         }
00305 
00306         public Rectangle GetBounds()
00307         {
00308             Rectangle bounds = this.Content.GetBounds();
00309 
00310             return new Rectangle(bounds.Location.X - this.Filter.TopLeftMargin.X, bounds.Location.Y -
      this.Filter.TopLeftMargin.Y, bounds.Size.Width + this.Filter.TopLeftMargin.X +
      this.Filter.BottomRightMargin.X, bounds.Size.Height + this.Filter.TopLeftMargin.Y +
      this.Filter.BottomRightMargin.Y);
00311         }
00312     }
00313 }
```

8.69 GraphicsPath.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents a graphics path that can be filled or stroked.
00026 /// </summary>
00027     public class GraphicsPath
00028     {
00029 /// <summary>
00030 /// The segments that make up the path.
00031 /// </summary>
00032         public List<Segment> Segments { get; set; } = new List<Segment>();
00033 
00034 
00035 /// <summary>
00036 /// Move the current point without tracing a segment from the previous point.
00037 /// </summary>
00038 /// <param name="p">The new point.</param>
00039 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00040         public GraphicsPath MoveTo(Point p)
00041         {
00042             cachedLength = double.NaN;
00043             cachedBounds = Rectangle.NaN;
```

```
00044             Segments.Add(new MoveSegment(p));
00045             return this;
00046         }
00047 
00048 /// <summary>
00049 /// Move the current point without tracing a segment from the previous point.
00050 /// </summary>
00051 /// <param name="x">The horizontal coordinate of the new point.</param>
00052 /// <param name="y">The vertical coordinate of the new point.</param>
00053 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00054         public GraphicsPath MoveTo(double x, double y)
00055         {
00056             MoveTo(new Point(x, y));
00057             return this;
00058         }
00059 
00060 /// <summary>
00061 /// Move the current point and trace a segment from the previous point.
00062 /// </summary>
00063 /// <param name="p">The new point.</param>
00064 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00065         public GraphicsPath LineTo(Point p)
00066         {
00067             cachedLength = double.NaN;
00068             cachedBounds = Rectangle.NaN;
00069 
00070             if (Segments.Count == 0)
00071             {
00072                 Segments.Add(new MoveSegment(p));
00073             }
00074             else
00075             {
00076                 Segments.Add(new LineSegment(p));
00077             }
00078             return this;
00079         }
00080 
00081 /// <summary>
00082 /// Move the current point and trace a segment from the previous point.
00083 /// </summary>
00084 /// <param name="x">The horizontal coordinate of the new point.</param>
00085 /// <param name="y">The vertical coordinate of the new point.</param>
00086 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00087         public GraphicsPath LineTo(double x, double y)
00088         {
00089             LineTo(new Point(x, y));
00090             return this;
00091         }
00092 
00093 /// <summary>
00094 /// Trace an arc segment from a circle with the specified <paramref name="center"/> and <paramref
      name="radius"/>, starting at <paramref name="startAngle"/> and ending at <paramref name="endAngle"/>.
00095 /// The current point is updated to the end point of the arc.
00096 /// </summary>
00097 /// <param name="center">The center of the arc.</param>
00098 /// <param name="radius">The radius of the arc.</param>
00099 /// <param name="startAngle">The start angle (in radians) of the arc.</param>
00100 /// <param name="endAngle">The end angle (in radians) of the arc.</param>
00101 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00102         public GraphicsPath Arc(Point center, double radius, double startAngle, double endAngle)
00103         {
00104             cachedLength = double.NaN;
00105             cachedBounds = Rectangle.NaN;
00106 
00107             if (Segments.Count == 0)
00108             {
00109                 Segments.Add(new MoveSegment(center.X + radius * Math.Cos(startAngle), center.Y +
      radius * Math.Sin(startAngle)));
00110             }
00111             Segments.Add(new ArcSegment(center, radius, startAngle, endAngle));
00112             return this;
00113         }
00114 
00115 /// <summary>
00116 /// Trace an arc segment from a circle with the specified center and <paramref name="radius"/>,
      starting at <paramref name="startAngle"/> and ending at <paramref name="endAngle"/>.
00117 /// The current point is updated to the end point of the arc.
00118 /// </summary>
00119 /// <param name="centerX">The horizontal coordinate of the center of the arc.</param>
00120 /// <param name="centerY">The vertical coordinate of the center of the arc.</param>
00121 /// <param name="radius">The radius of the arc.</param>
00122 /// <param name="startAngle">The start angle (in radians) of the arc.</param>
00123 /// <param name="endAngle">The end angle (in radians) of the arc.</param>
00124 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00125         public GraphicsPath Arc(double centerX, double centerY, double radius, double startAngle,
      double endAngle)
00126         {
```

```
00127             Arc(new Point(centerX, centerY), radius, startAngle, endAngle);
00128             return this;
00129         }
00130 
00131 /// <summary>
00132 /// Trace an arc from an ellipse with the specified radii, rotated by <paramref name="axisAngle"/>
      with respect to the x-axis, starting at the current point and ending at the <paramref
      name="endPoint"/>.
00133 /// </summary>
00134 /// <param name="radiusX">The horizontal radius of the ellipse.</param>
00135 /// <param name="radiusY">The vertical radius of the ellipse.</param>
00136 /// <param name="axisAngle">The angle of the horizontal axis of the ellipse with respect to the
      horizontal axis.</param>
00137 /// <param name="largeArc">Determines whether the large or the small arc is drawn.</param>
00138 /// <param name="sweepClockwise">Determines whether the clockwise or anticlockwise arc is
      drawn.</param>
00139 /// <param name="endPoint">The end point of the arc.</param>
00140 /// <returns></returns>
00141         public GraphicsPath EllipticalArc(double radiusX, double radiusY, double axisAngle, bool
      largeArc, bool sweepClockwise, Point endPoint)
00142         {
00143             if (radiusX == 0 || radiusY == 0)
00144             {
00145                 return this.LineTo(endPoint);
00146             }
00147             else
00148             {
00149                 radiusX = Math.Abs(radiusX);
00150                 radiusY = Math.Abs(radiusY);
00151             }
00152 
00153             double x1 = 0;
00154             double y1 = 0;
00155 
00156             if (this.Segments.Count > 0)
00157             {
00158                 for (int i = this.Segments.Count - 1; i >= 0; i--)
00159                 {
00160                     if (this.Segments[i].Type != SegmentType.Close)
00161                     {
00162                         x1 = this.Segments[i].Point.X;
00163                         y1 = this.Segments[i].Point.Y;
00164                         break;
00165                     }
00166                 }
00167             }
00168 
00169             double x2 = endPoint.X;
00170             double y2 = endPoint.Y;
00171 
00172             double x1P = Math.Cos(axisAngle) * (x1 - x2) * 0.5 + Math.Sin(axisAngle) * (y1 - y2) *
      0.5;
00173 
00174             if (Math.Abs(x1P) < 1e-7)
00175             {
00176                 x1P = 0;
00177             }
00178 
00179             double y1P = -Math.Sin(axisAngle) * (x1 - x2) * 0.5 + Math.Cos(axisAngle) * (y1 - y2) *
      0.5;
00180 
00181             if (Math.Abs(y1P) < 1e-7)
00182             {
00183                 y1P = 0;
00184             }
00185 
00186             double lambda = x1P * x1P / (radiusX * radiusX) + y1P * y1P / (radiusY * radiusY);
00187 
00188             if (lambda > 1)
00189             {
00190                 double sqrtLambda = Math.Sqrt(lambda);
00191                 radiusX *= sqrtLambda;
00192                 radiusY *= sqrtLambda;
00193             }
00194 
00195             double sqrtTerm = (largeArc != sweepClockwise ?  1 :  -1) * Math.Sqrt(Math.Max(0, (radiusX
      * radiusX * radiusY * radiusY - radiusX * radiusX * y1P * y1P - radiusY * radiusY * x1P * x1P) /
      (radiusX * radiusX * y1P * y1P + radiusY * radiusY * x1P * x1P)));
00196 
00197             double cXP = sqrtTerm * radiusX * y1P / radiusY;
00198             double cYP = -sqrtTerm * radiusY * x1P / radiusX;
00199 
00200             double cX = Math.Cos(axisAngle) * cXP - Math.Sin(axisAngle) * cYP + (x1 + x2) * 0.5;
00201             double cY = Math.Sin(axisAngle) * cXP + Math.Cos(axisAngle) * cYP + (y1 + y2) * 0.5;
00202 
00203             double theta1 = AngleVectors(1, 0, (x1P - cXP) / radiusX, (y1P - cYP) / radiusY);
00204             double deltaTheta = AngleVectors((x1P - cXP) / radiusX, (y1P - cYP) / radiusY, (-x1P -
```

```
      cXP) / radiusX, (-y1P - cYP) / radiusY) % (2 * Math.PI);
00205
00206             if (!sweepClockwise && deltaTheta > 0)
00207             {
00208                 deltaTheta -= 2 * Math.PI;
00209             }
00210             else if (sweepClockwise && deltaTheta < 0)
00211             {
00212                 deltaTheta += 2 * Math.PI;
00213             }
00214
00215             double r = Math.Min(radiusX, radiusY);
00216
00217             ArcSegment arc = new ArcSegment(0, 0, r, theta1, theta1 + deltaTheta);
00218
00219             Segment[] segments = arc.ToBezierSegments();
00220
00221             for (int i = 0; i < segments.Length; i++)
00222             {
00223                 for (int j = 0; j < segments[i].Points.Length; j++)
00224                 {
00225                     double newX = segments[i].Points[j].X * radiusX / r;
00226                     double newY = segments[i].Points[j].Y * radiusY / r;
00227
00228                     segments[i].Points[j] = new Point(newX * Math.Cos(axisAngle) - newY *
      Math.Sin(axisAngle) + cX, newX * Math.Sin(axisAngle) + newY * Math.Cos(axisAngle) + cY);
00229                 }
00230             }
00231
00232             cachedLength = double.NaN;
00233             cachedBounds = Rectangle.NaN;
00234
00235             this.Segments.AddRange(segments);
00236
00237             return this;
00238         }
00239
00240         private static double AngleVectors(double uX, double uY, double vX, double vY)
00241         {
00242             double tbr = Math.Acos((uX * vX + uY * vY) / Math.Sqrt((uX * uX + uY * uY) * (vX * vX + vY
      * vY)));
00243             double sign = Math.Sign(uX * vY - uY * vX);
00244             if (sign != 0)
00245             {
00246                 tbr *= sign;
00247             }
00248             return tbr;
00249         }
00250
00251
00252 /// <summary>
00253 /// Trace a cubic Bezier curve from the current point to a destination point, with two control points.
00254 /// The current point is updated to the end point of the Bezier curve.
00255 /// </summary>
00256 /// <param name="control1">The first control point.</param>
00257 /// <param name="control2">The second control point.</param>
00258 /// <param name="endPoint">The destination point.</param>
00259 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00260         public GraphicsPath CubicBezierTo(Point control1, Point control2, Point endPoint)
00261         {
00262             cachedLength = double.NaN;
00263             cachedBounds = Rectangle.NaN;
00264
00265             if (Segments.Count == 0)
00266             {
00267                 Segments.Add(new MoveSegment(control1));
00268             }
00269             Segments.Add(new CubicBezierSegment(control1, control2, endPoint));
00270             return this;
00271         }
00272
00273 /// <summary>
00274 /// Trace a cubic Bezier curve from the current point to a destination point, with two control points.
00275 /// The current point is updated to the end point of the Bezier curve.
00276 /// </summary>
00277 /// <param name="control1X">The horizontal coordinate of the first control point.</param>
00278 /// <param name="control1Y">The vertical coordinate of the first control point.</param>
00279 /// <param name="control2X">The horizontal coordinate of the second control point.</param>
00280 /// <param name="control2Y">The vertical coordinate of the second control point.</param>
00281 /// <param name="endPointX">The horizontal coordinate of the destination point.</param>
00282 /// <param name="endPointY">The vertical coordinate of the destination point.</param>
00283 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00284         public GraphicsPath CubicBezierTo(double control1X, double control1Y, double control2X, double
      control2Y, double endPointX, double endPointY)
00285         {
00286             CubicBezierTo(new Point(control1X, control1Y), new Point(control2X, control2Y), new
      Point(endPointX, endPointY));
```

```
00287             return this;
00288         }
00289 
00290 /// <summary>
00291 /// Trace a quadratic Bezier curve from the current point to a destination point, with a single
      control point.
00292 /// The current point is updated to the end point of the Bezier curve.
00293 /// </summary>
00294 /// <param name="control">The control point.</param>
00295 /// <param name="endPoint">The destination point.</param>
00296 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00297         public GraphicsPath QuadraticBezierTo(Point control, Point endPoint)
00298         {
00299             Point currentPoint = control;
00300 
00301             if (Segments.Count > 0)
00302             {
00303                 for (int i = Segments.Count - 1; i >= 0; i--)
00304                 {
00305                     if (Segments[i].Type != SegmentType.Close)
00306                     {
00307                         currentPoint = Segments[i].Point;
00308                         break;
00309                     }
00310                 }
00311             }
00312 
00313             Point control1 = new Point((currentPoint.X + 2 * control.X) / 3, (currentPoint.Y + 2 *
      control.Y) / 3);
00314             Point control2 = new Point((endPoint.X + 2 * control.X) / 3, (endPoint.Y + 2 * control.Y)
      / 3);
00315 
00316             CubicBezierTo(control1, control2, endPoint);
00317 
00318             return this;
00319         }
00320 
00321 /// <summary>
00322 /// Trace a quadratic Bezier curve from the current point to a destination point, with a single
      control point.
00323 /// The current point is updated to the end point of the Bezier curve.
00324 /// </summary>
00325 /// <param name="controlX">The horizontal coordinate of the control point.</param>
00326 /// <param name="controlY">The vertical coordinate of the control point.</param>
00327 /// <param name="endPointX">The horizontal coordinate of the destination point.</param>
00328 /// <param name="endPointY">The vertical coordinate of the destination point.</param>
00329 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00330         public GraphicsPath QuadraticBezierTo(double controlX, double controlY, double endPointX,
      double endPointY)
00331         {
00332             QuadraticBezierTo(new Point(controlX, controlY), new Point(endPointX, endPointY));
00333             return this;
00334         }
00335 
00336 /// <summary>
00337 /// Trace a segment from the current point to the start point of the figure and flag the figure as
      closed.
00338 /// </summary>
00339 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00340         public GraphicsPath Close()
00341         {
00342             cachedLength = double.NaN;
00343             cachedBounds = Rectangle.NaN;
00344             Segments.Add(new CloseSegment());
00345             return this;
00346         }
00347 
00348 /// <summary>
00349 /// Add the contour of a text string to the current path.
00350 /// </summary>
00351 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00352 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00353 /// <param name="text">The string to draw.</param>
00354 /// <param name="font">The font with which to draw the text.</param>
00355 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00356 /// /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00357         public GraphicsPath AddText(double originX, double originY, string text, Font font,
      TextBaselines textBaseline = TextBaselines.Top)
00358         {
00359             return AddText(new Point(originX, originY), text, font, textBaseline);
00360         }
00361 
00362 /// <summary>
00363 /// Add the contour of a text string to the current path.
00364 /// </summary>
```

```
00365 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00366 /// <param name="text">The string to draw.</param>
00367 /// <param name="font">The font with which to draw the text.</param>
00368 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00369 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00370         public GraphicsPath AddText(Point origin, string text, Font font, TextBaselines textBaseline =
      TextBaselines.Top)
00371         {
00372             if (!string.IsNullOrEmpty(text))
00373             {
00374                 Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(text);
00375 
00376                 Point baselineOrigin = origin;
00377 
00378                 switch (textBaseline)
00379                 {
00380                     case TextBaselines.Baseline:
00381                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y);
00382                         break;
00383                     case TextBaselines.Top:
00384                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Top);
00385                         break;
00386                     case TextBaselines.Bottom:
00387                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Bottom);
00388                         break;
00389                     case TextBaselines.Middle:
00390                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      (metrics.Top - metrics.Bottom) * 0.5 + metrics.Bottom);
00391                         break;
00392                 }
00393 
00394                 Point currentGlyphPlacementDelta = new Point();
00395                 Point currentGlyphAdvanceDelta = new Point();
00396                 Point nextGlyphPlacementDelta = new Point();
00397                 Point nextGlyphAdvanceDelta = new Point();
00398 
00399                 for (int i = 0; i < text.Length; i++)
00400                 {
00401                     char c = text[i];
00402 
00403                     if (Font.EnableKerning && i < text.Length - 1)
00404                     {
00405                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00406                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00407                         nextGlyphAdvanceDelta = new Point();
00408                         nextGlyphPlacementDelta = new Point();
00409 
00410                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00411 
00412                         if (kerning != null)
00413                         {
00414                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00415                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00416 
00417                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00418                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00419                         }
00420                     }
00421 
00422                     TrueTypeFile.TrueTypePoint[][] glyphPaths =
      font.FontFamily.TrueTypeFile.GetGlyphPath(c, font.FontSize);
00423 
00424                     for (int j = 0; j < glyphPaths.Length; j++)
00425                     {
00426                         for (int k = 0; k < glyphPaths[j].Length; k++)
00427                         {
00428                             if (k == 0)
00429                             {
00430                                 this.MoveTo(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00431                             }
00432                             else
00433                             {
00434                                 if (glyphPaths[j][k].IsOnCurve)
00435                                 {
00436                                     this.LineTo(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00437                                 }
00438                                 else
```

```
00439                                 {
00440                                     Point startPoint = this.Segments.Last().Point;
00441                                     Point quadCtrl = new Point(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00442                                     Point endPoint = new Point(glyphPaths[j][k + 1].X +
      baselineOrigin.X + currentGlyphPlacementDelta.X, -glyphPaths[j][k + 1].Y + baselineOrigin.Y +
      currentGlyphPlacementDelta.Y);
00443
00444
00445                                     Point ctrl1 = new Point(startPoint.X / 3 + 2 * quadCtrl.X / 3,
      startPoint.Y / 3 + 2 * quadCtrl.Y / 3);
00446                                     Point ctrl2 = new Point(endPoint.X / 3 + 2 * quadCtrl.X / 3,
      endPoint.Y / 3 + 2 * quadCtrl.Y / 3);
00447
00448                                     this.CubicBezierTo(ctrl1, ctrl2, endPoint);
00449
00450                                     k++;
00451                                 }
00452                             }
00453                         }
00454
00455                         this.Close();
00456                     }
00457
00458                     baselineOrigin.X += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00459                     baselineOrigin.Y += (currentGlyphAdvanceDelta.Y) * font.FontSize / 1000;
00460                 }
00461             }
00462             return this;
00463         }
00464
00465 /// <summary>
00466 /// Add the contour of a text string flowing along a <see cref="GraphicsPath"/> to the current path.
00467 /// </summary>
00468 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00469 /// <param name="text">The string to draw.</param>
00470 /// <param name="font">The font with which to draw the text.</param>
00471 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
00472 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00473 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00474 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00475         public GraphicsPath AddTextOnPath(GraphicsPath path, string text, Font font, double reference
      = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline = TextBaselines.Top)
00476         {
00477             if (!string.IsNullOrEmpty(text))
00478             {
00479                 cachedLength = double.NaN;
00480                 cachedBounds = Rectangle.NaN;
00481
00482                 double currDelta = 0;
00483                 double pathLength = path.MeasureLength();
00484
00485                 Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00486
00487                 switch (anchor)
00488                 {
00489                     case TextAnchors.Left:
00490                         break;
00491                     case TextAnchors.Center:
00492                         currDelta = -fullMetrics.Width * 0.5 / pathLength;
00493                         break;
00494                     case TextAnchors.Right:
00495                         currDelta = -fullMetrics.Width / pathLength;
00496                         break;
00497                 }
00498
00499                 Point currentGlyphPlacementDelta = new Point();
00500                 Point currentGlyphAdvanceDelta = new Point();
00501                 Point nextGlyphPlacementDelta = new Point();
00502                 Point nextGlyphAdvanceDelta = new Point();
00503
00504                 for (int i = 0; i < text.Length; i++)
00505                 {
00506                     string c = text.Substring(i, 1);
00507
00508                     if (Font.EnableKerning && i < text.Length - 1)
00509                     {
00510                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00511                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00512                         nextGlyphAdvanceDelta = new Point();
00513                         nextGlyphPlacementDelta = new Point();
00514
00515                         TrueTypeFile.PairKerning kerning =
```

```
font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00516
00517                         if (kerning != null)
00518                         {
00519                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00520                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00521
00522                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00523                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00524                         }
00525                     }
00526
00527                     Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00528
00529                     Point origin = path.GetPointAtRelative(reference + currDelta +
currentGlyphPlacementDelta.X * font.FontSize / 1000);
00530
00531                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
metrics.LeftSideBearing) / pathLength * 0.5);
00532
00533                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00534
00535                     GraphicsPath glyphPath = new GraphicsPath();
00536
00537                     switch (textBaseline)
00538                     {
00539                         case TextBaselines.Top:
00540                             if (i > 0)
00541                             {
00542                                 glyphPath.AddText(new Point(metrics.LeftSideBearing, fullMetrics.Top),
c, font, textBaseline:  TextBaselines.Baseline);
00543                             }
00544                             else
00545                             {
00546                                 glyphPath.AddText(new Point(0, fullMetrics.Top), c, font,
textBaseline:  TextBaselines.Baseline);
00547                             }
00548                             break;
00549                         case TextBaselines.Baseline:
00550                             if (i > 0)
00551                             {
00552                                 glyphPath.AddText(new Point(metrics.LeftSideBearing, 0), c, font,
textBaseline:  TextBaselines.Baseline);
00553                             }
00554                             else
00555                             {
00556                                 glyphPath.AddText(new Point(0, 0), c, font, textBaseline:
TextBaselines.Baseline);
00557                             }
00558                             break;
00559                         case TextBaselines.Bottom:
00560                             if (i > 0)
00561                             {
00562                                 glyphPath.AddText(new Point(metrics.LeftSideBearing,
fullMetrics.Bottom), c, font, textBaseline:  TextBaselines.Baseline);
00563                             }
00564                             else
00565                             {
00566                                 glyphPath.AddText(new Point(0, fullMetrics.Bottom), c, font,
textBaseline:  TextBaselines.Baseline);
00567                             }
00568                             break;
00569                         case TextBaselines.Middle:
00570                             if (i > 0)
00571                             {
00572                                 glyphPath.AddText(new Point(metrics.LeftSideBearing,
fullMetrics.Bottom + fullMetrics.Height / 2), c, font, textBaseline:  TextBaselines.Baseline);
00573                             }
00574                             else
00575                             {
00576                                 glyphPath.AddText(new Point(0, fullMetrics.Bottom + fullMetrics.Height
/ 2), c, font, textBaseline:  TextBaselines.Baseline);
00577                             }
00578                             break;
00579                     }
00580
00581                     double angle = Math.Atan2(tangent.Y, tangent.X);
00582
00583                     for (int j = 0; j < glyphPath.Segments.Count; j++)
00584                     {
00585                         if (glyphPath.Segments[j].Points != null)
```

```
00586                         {
00587                             for (int k = 0; k < glyphPath.Segments[j].Points.Length; k++)
00588                             {
00589                                 double newX = glyphPath.Segments[j].Points[k].X * Math.Cos(angle) -
      glyphPath.Segments[j].Points[k].Y * Math.Sin(angle) + origin.X;
00590                                 double newY = glyphPath.Segments[j].Points[k].X * Math.Sin(angle) +
      glyphPath.Segments[j].Points[k].Y * Math.Cos(angle) + origin.Y;
00591 
00592                                 glyphPath.Segments[j].Points[k] = new Point(newX, newY);
00593                             }
00594                         }
00595 
00596                         this.Segments.Add(glyphPath.Segments[j]);
00597                     }
00598 
00599                     if (i > 0)
00600                     {
00601                         currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00602                     }
00603                     else
00604                     {
00605                         currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00606                     }
00607                 }
00608             }
00609 
00610             return this;
00611         }
00612 
00613 
00614 
00615 /// <summary>
00616 /// Add the contour of the underline of the specified text string to the current path.
00617 /// </summary>
00618 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00619 /// <param name="text">The string whose underline will be drawn.</param>
00620 /// <param name="font">The font with which to draw the text.</param>
00621 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00622 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00623         public GraphicsPath AddTextUnderline(Point origin, string text, Font font, TextBaselines
      textBaseline = TextBaselines.Top)
00624         {
00625             if (!string.IsNullOrEmpty(text))
00626             {
00627                 if (font.Underline == null)
00628                 {
00629                     return this;
00630                 }
00631 
00632                 Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(text);
00633 
00634                 double italicAngle = font.FontFamily.TrueTypeFile?.GetItalicAngle() ??  0;
00635 
00636                 if (double.IsNaN(italicAngle))
00637                 {
00638                     italicAngle = 0;
00639                 }
00640 
00641                 Point baselineOrigin = origin;
00642 
00643                 switch (textBaseline)
00644                 {
00645                     case TextBaselines.Baseline:
00646                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y);
00647                         break;
00648                     case TextBaselines.Top:
00649                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Top);
00650                         break;
00651                     case TextBaselines.Bottom:
00652                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Bottom);
00653                         break;
00654                     case TextBaselines.Middle:
00655                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      (metrics.Top - metrics.Bottom) * 0.5 + metrics.Bottom);
00656                         break;
00657                 }
00658 
00659                 if (!font.Underline.SkipDescenders)
00660                 {
00661                     double italicShift;
00662 
00663                     if (!font.Underline.FollowItalicAngle || italicAngle == 0)
```

```
00664                     {
00665                         italicShift = 0;
00666                     }
00667                     else
00668                     {
00669                         italicShift = font.Underline.Thickness * font.FontSize * Math.Tan(italicAngle
      / 180.0 * Math.PI);
00670                     }
00671 
00672                     if (font.Underline.LineCap == LineCaps.Butt)
00673                     {
00674                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00675                         {
00676                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00677                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00678                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
00679                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00680                             this.Close();
00681                         }
00682                         else
00683                         {
00684                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00685                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00686                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
00687                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
00688                             this.Close();
00689                         }
00690                     }
00691                     else if (font.Underline.LineCap == LineCaps.Square)
00692                     {
00693                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00694                         {
00695                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
00696                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
00697                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize);
00698                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize);
00699                             this.Close();
00700                         }
00701                         else
00702                         {
00703                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
00704                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
00705                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      font.Underline.Thickness * font.FontSize * 0.5 + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00706                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize * 0.5 + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00707                             this.Close();
00708                         }
00709                     }
00710                     else if (font.Underline.LineCap == LineCaps.Round)
00711                     {
00712                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00713                         {
00714                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00715                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00716                             this.Arc(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize
      * 0.5, font.Underline.Thickness * font.FontSize * 0.5, -Math.PI / 2, Math.PI / 2);
00717                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
```

```
00718                           this.Arc(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize * 0.5,
      font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
00719                           this.Close();
00720                       }
00721                       else
00722                       {
00723                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing - italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00724                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00725                           this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing +
      metrics.Width + font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
      font.FontSize,
00726                               baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      italicShift + font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize,
00727                               baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
      italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
00728
00729                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00730                           this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize,
00731                               baselineOrigin.X + metrics.LeftSideBearing - italicShift -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
00732                               baselineOrigin.X + metrics.LeftSideBearing - italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00733
00734                           this.Close();
00735                       }
00736                   }
00737
00738                   return this;
00739               }
00740               else
00741               {
00742                   if (font.Underline.LineCap == LineCaps.Butt)
00743                   {
00744                       double italicShift;
00745
00746                       if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00747                       {
00748                           italicShift = 0;
00749                       }
00750                       else
00751                       {
00752                           italicShift = font.Underline.Thickness * font.FontSize *
      Math.Tan(italicAngle / 180.0 * Math.PI);
00753                       }
00754
00755                       bool started = false;
00756
00757                       double currX = baselineOrigin.X;
00758                       double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00759                       double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00760
00761                       Point currentGlyphPlacementDelta = new Point();
00762                       Point currentGlyphAdvanceDelta = new Point();
00763                       Point nextGlyphPlacementDelta = new Point();
00764                       Point nextGlyphAdvanceDelta = new Point();
00765
00766                       for (int i = 0; i < text.Length; i++)
00767                       {
00768                           char c = text[i];
00769
00770                           if (Font.EnableKerning && i < text.Length - 1)
00771                           {
00772                               currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00773                               currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00774                               nextGlyphAdvanceDelta = new Point();
00775                               nextGlyphPlacementDelta = new Point();
00776
00777                               TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00778
00779                               if (kerning != null)
00780                               {
00781                                   currentGlyphPlacementDelta = new
      Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
      kerning.Glyph1Placement.Y);
00782                                   currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00783
00784                                   nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
```

```
kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00785                                   nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00786                               }
00787                           }
00788
00789                           double[] intersections =
font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
font.Underline.Thickness * 1000);
00790
00791                           if (intersections != null)
00792                           {
00793                               intersections[0] = intersections[0] * font.FontSize / 1000;
00794                               intersections[1] = intersections[1] * font.FontSize / 1000;
00795
00796                               if (currX + intersections[0] - font.Underline.Thickness *
font.FontSize >= underlineStartX)
00797                               {
00798                                   if (!started)
00799                                   {
00800                                       started = true;
00801                                       this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing +
italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
font.Underline.Position * font.FontSize);
00802                                   }
00803
00804                                   this.LineTo(currX + intersections[0] - font.Underline.Thickness *
font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00805                                   this.LineTo(currX + intersections[0] - font.Underline.Thickness *
font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
font.Underline.Thickness * font.FontSize);
00806                                   this.Close();
00807                               }
00808
00809                               started = true;
00810
00811                               this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
font.Underline.Thickness * font.FontSize);
00812                               this.LineTo(currX + intersections[1] + font.Underline.Thickness *
font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00813
00814                               underlineStartX = currX + intersections[1] + font.Underline.Thickness
* font.FontSize;
00815                               currUnderlineX = Math.Max(currX + intersections[1] +
font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
+ currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
00816                           }
00817                           else if (i == text.Length - 1)
00818                           {
00819                               if (c != ' ')
00820                               {
00821                                   currUnderlineX +=
(font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
00822                               }
00823                               else
00824                               {
00825                                   currUnderlineX +=
(font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
1000;
00826                               }
00827                           }
00828                           else
00829                           {
00830                               currUnderlineX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
+ currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00831                           }
00832
00833                           currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00834                       }
00835
00836                       if (!started)
00837                       {
00838                           started = true;
00839                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift,
baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
font.Underline.Position * font.FontSize);
00840                       }
00841
00842                       this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
font.FontSize);
00843                       this.LineTo(currUnderlineX + italicShift, baselineOrigin.Y +
```

```
font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00844                         this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00845                         this.Close();
00846                     }
00847                     else if (font.Underline.LineCap == LineCaps.Square)
00848                     {
00849                         double italicShift;
00850 
00851                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00852                         {
00853                             italicShift = 0;
00854                         }
00855                         else
00856                         {
00857                             italicShift = font.Underline.Thickness * font.FontSize *
Math.Tan(italicAngle / 180.0 * Math.PI);
00858                         }
00859 
00860                         bool started = false;
00861 
00862                         double currX = baselineOrigin.X;
00863                         double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00864                         double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00865 
00866                         Point currentGlyphPlacementDelta = new Point();
00867                         Point currentGlyphAdvanceDelta = new Point();
00868                         Point nextGlyphPlacementDelta = new Point();
00869                         Point nextGlyphAdvanceDelta = new Point();
00870 
00871                         for (int i = 0; i < text.Length; i++)
00872                         {
00873                             char c = text[i];
00874 
00875                             if (Font.EnableKerning && i < text.Length - 1)
00876                             {
00877                                 currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00878                                 currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00879                                 nextGlyphAdvanceDelta = new Point();
00880                                 nextGlyphPlacementDelta = new Point();
00881 
00882                                 TrueTypeFile.PairKerning kerning =
font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00883 
00884                                 if (kerning != null)
00885                                 {
00886                                     currentGlyphPlacementDelta = new
Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
kerning.Glyph1Placement.Y);
00887                                     currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00888 
00889                                     nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00890                                     nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00891                                 }
00892                             }
00893 
00894                             double[] intersections =
font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
font.Underline.Thickness * 1000);
00895 
00896                             if (intersections != null)
00897                             {
00898                                 intersections[0] = intersections[0] * font.FontSize / 1000;
00899                                 intersections[1] = intersections[1] * font.FontSize / 1000;
00900 
00901                                 if (currX + intersections[0] - font.Underline.Thickness *
font.FontSize >= underlineStartX)
00902                                 {
00903                                     if (!started)
00904                                     {
00905                                         started = true;
00906                                         this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing +
italicShift - font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y +
font.Underline.Position * font.FontSize + font.Underline.Thickness *
font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing - font.Underline.Thickness *
font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00907                                     }
00908                                     this.LineTo(currX + intersections[0] - font.Underline.Thickness *
font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00909                                     this.LineTo(currX + intersections[0] - font.Underline.Thickness *
font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
font.Underline.Thickness * font.FontSize);
00910 
00911                                     this.Close();
```

```
00912                                 }
00913
00914                                 started = true;
00915
00916                                 this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00917                                 this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00918
00919                                 underlineStartX = currX + intersections[1] + font.Underline.Thickness
      * font.FontSize;
00920                                 currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
00921                             }
00922                             else if (i == text.Length - 1)
00923                             {
00924                                 if (c != ' ')
00925                                 {
00926                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
00927                                 }
00928                                 else
00929                                 {
00930                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
00931                                 }
00932                             }
00933                             else
00934                             {
00935                                 currUnderlineX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00936                             }
00937
00938                             currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00939                         }
00940
00941                         if (!started)
00942                         {
00943                             started = true;
00944                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift -
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize).LineTo(baselineOrigin.X +
      metrics.LeftSideBearing - font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00945                         }
00946
00947                         this.LineTo(currUnderlineX + font.Underline.Thickness * font.FontSize * 0.5,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00948                         this.LineTo(currUnderlineX + italicShift + font.Underline.Thickness *
      font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00949                         this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00950                         this.Close();
00951                     }
00952                     else if (font.Underline.LineCap == LineCaps.Round)
00953                     {
00954                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00955                         {
00956                             bool started = false;
00957
00958                             double currX = baselineOrigin.X;
00959                             double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00960                             double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00961
00962                             Point currentGlyphPlacementDelta = new Point();
00963                             Point currentGlyphAdvanceDelta = new Point();
00964                             Point nextGlyphPlacementDelta = new Point();
00965                             Point nextGlyphAdvanceDelta = new Point();
00966
00967                             for (int i = 0; i < text.Length; i++)
00968                             {
00969                                 char c = text[i];
00970
00971                                 if (Font.EnableKerning && i < text.Length - 1)
00972                                 {
00973                                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00974                                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00975                                     nextGlyphAdvanceDelta = new Point();
00976                                     nextGlyphPlacementDelta = new Point();
00977
```

```
00978                                   TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00979
00980                                   if (kerning != null)
00981                                   {
00982                                       currentGlyphPlacementDelta = new
      Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
      kerning.Glyph1Placement.Y);
00983                                       currentGlyphAdvanceDelta = new
      Point(currentGlyphAdvanceDelta.X + kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y +
      kerning.Glyph1Advance.Y);
00984
00985                                       nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X
      + kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00986                                       nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00987                                   }
00988                               }
00989
00990                               double[] intersections =
      font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
      font.Underline.Thickness * 1000);
00991
00992                               if (intersections != null)
00993                               {
00994                                   intersections[0] = intersections[0] * font.FontSize / 1000;
00995                                   intersections[1] = intersections[1] * font.FontSize / 1000;
00996
00997                                   if (currX + intersections[0] - font.Underline.Thickness *
      font.FontSize >= underlineStartX)
00998                                   {
00999                                       if (!started)
01000                                       {
01001                                           started = true;
01002                                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01003                                           this.Arc(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize
      * 0.5, font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
01004                                       }
01005
01006                                       this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01007                                       this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01008
01009                                       this.Close();
01010                                   }
01011
01012                                   started = true;
01013
01014                                   this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01015                                   this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01016
01017                                   underlineStartX = currX + intersections[1] +
      font.Underline.Thickness * font.FontSize;
01018                                   currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
01019                               }
01020                               else if (i == text.Length - 1)
01021                               {
01022                                   if (c != ' ')
01023                                   {
01024                                       currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
01025                                   }
01026                                   else
01027                                   {
01028                                       currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01029                                   }
01030                               }
01031                               else
01032                               {
01033                                   currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01034                               }
```

```
01035
01036                               currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
01037                               }
01038
01039                               if (!started)
01040                               {
01041                                   started = true;
01042                                   this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01043                                   this.Arc(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y
      + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize * 0.5,
      font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
01044                               }
01045
01046                               this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
01047
01048                               this.Arc(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize * 0.5, font.Underline.Thickness *
      font.FontSize * 0.5, -Math.PI / 2, Math.PI / 2);
01049
01050                               this.LineTo(underlineStartX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize);
01051                               this.Close();
01052                           }
01053                           else
01054                           {
01055                               double italicShift = font.Underline.Thickness * font.FontSize *
      Math.Tan(italicAngle / 180.0 * Math.PI);
01056
01057                               bool started = false;
01058
01059                               double currX = baselineOrigin.X;
01060                               double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
01061                               double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
01062
01063                               Point currentGlyphPlacementDelta = new Point();
01064                               Point currentGlyphAdvanceDelta = new Point();
01065                               Point nextGlyphPlacementDelta = new Point();
01066                               Point nextGlyphAdvanceDelta = new Point();
01067
01068                               for (int i = 0; i < text.Length; i++)
01069                               {
01070                                   char c = text[i];
01071
01072                                   if (Font.EnableKerning && i < text.Length - 1)
01073                                   {
01074                                       currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01075                                       currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01076                                       nextGlyphAdvanceDelta = new Point();
01077                                       nextGlyphPlacementDelta = new Point();
01078
01079                                       TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
01080
01081                                       if (kerning != null)
01082                                       {
01083                                           currentGlyphPlacementDelta = new
      Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
      kerning.Glyph1Placement.Y);
01084                                           currentGlyphAdvanceDelta = new
      Point(currentGlyphAdvanceDelta.X + kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y +
      kerning.Glyph1Advance.Y);
01085
01086                                           nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X
      + kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01087                                           nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01088                                       }
01089                                   }
01090
01091                                   double[] intersections =
      font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
      font.Underline.Thickness * 1000);
01092
01093                                   if (intersections != null)
01094                                   {
01095                                       intersections[0] = intersections[0] * font.FontSize / 1000;
01096                                       intersections[1] = intersections[1] * font.FontSize / 1000;
01097
01098                                       if (currX + intersections[0] - font.Underline.Thickness *
      font.FontSize >= underlineStartX)
01099                                       {
01100                                           if (!started)
01101                                           {
```

```
01102                                           started = true;
01103
01104                                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01105
01106                                           this.CubicBezierTo(baselineOrigin.X +
      metrics.LeftSideBearing - font.Underline.Thickness * font.FontSize, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize,
01107                                                   baselineOrigin.X + metrics.LeftSideBearing -
      italicShift - font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
      font.FontSize,
01108                                                   baselineOrigin.X + metrics.LeftSideBearing -
      italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01109                                       }
01110
01111                                       this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01112                                       this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize);
01113
01114                                       this.Close();
01115                                   }
01116
01117                                   started = true;
01118
01119                                   this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01120                                   this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01121
01122                                   underlineStartX = currX + intersections[1] +
      font.Underline.Thickness * font.FontSize;
01123                                   currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
01124                               }
01125                               else if (i == text.Length - 1)
01126                               {
01127                                   if (c != ' ')
01128                                   {
01129                                       currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
01130                                   }
01131                                   else
01132                                   {
01133                                       currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01134                                   }
01135                               }
01136                               else
01137                               {
01138                                   currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01139                               }
01140
01141                               currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
01142                           }
01143
01144                           if (!started)
01145                           {
01146                               started = true;
01147
01148                               this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01149
01150                               this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize,
01151                                   baselineOrigin.X + metrics.LeftSideBearing - italicShift -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
01152                                   baselineOrigin.X + metrics.LeftSideBearing - italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
01153                           }
01154
01155                           this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
01156
01157                           //this.LineTo(currUnderlineX + italicShift + font.Underline.Thickness *
```

```
      font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01158                             this.CubicBezierTo(currUnderlineX + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
01159                                 currUnderlineX + italicShift + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize,
01160                                 currUnderlineX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
01161
01162                             this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
01163                             this.Close();
01164                         }
01165                     }
01166                 }
01167             }
01168             return this;
01169         }
01170
01171
01172 /// <summary>
01173 /// Adds a smooth spline composed of cubic bezier segments that pass through the specified points.
01174 /// </summary>
01175 /// <param name="points">The points through which the spline should pass.</param>
01176 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
01177         public GraphicsPath AddSmoothSpline(params Point[] points)
01178         {
01179             if (points.Length == 0)
01180             {
01181                 return this;
01182             }
01183             else if (points.Length == 1)
01184             {
01185                 return this.LineTo(points[0]);
01186             }
01187             else if (points.Length == 2)
01188             {
01189                 return this.LineTo(points[0]).LineTo(points[1]);
01190             }
01191
01192             Point[] smoothedSpline = SmoothSpline.SmoothSplines(points);
01193
01194             this.LineTo(smoothedSpline[0]);
01195
01196             for (int i = 1; i < smoothedSpline.Length; i += 3)
01197             {
01198                 this.CubicBezierTo(smoothedSpline[i], smoothedSpline[i + 1], smoothedSpline[i + 2]);
01199             }
01200
01201             return this;
01202         }
01203
01204 /// <summary>
01205 /// Adds another <see cref="GraphicsPath"/> to the current <see cref="GraphicsPath"/>.
01206 /// </summary>
01207 /// <param name="path">The existing <see cref="GraphicsPath"/> that should be added to the current
      <see cref="GraphicsPath"/>.</param>
01208 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
01209         public GraphicsPath AddPath(GraphicsPath path)
01210         {
01211             this.Segments.AddRange(path.Segments);
01212             return this;
01213         }
01214
01215         private double cachedLength = double.NaN;
01216
01217 /// <summary>
01218 /// Measures the length of the <see cref="GraphicsPath"/>.
01219 /// </summary>
01220 /// <returns>The length of the <see cref="GraphicsPath"/></returns>
01221         public double MeasureLength()
01222         {
01223             if (double.IsNaN(cachedLength))
01224             {
01225                 cachedLength = 0;
01226                 Point currPoint = new Point();
01227                 Point figureStartPoint = new Point();
01228
01229                 for (int i = 0; i < this.Segments.Count; i++)
01230                 {
01231                     switch (this.Segments[i].Type)
01232                     {
01233                         case SegmentType.Move:
01234                             currPoint = this.Segments[i].Point;
01235                             figureStartPoint = this.Segments[i].Point;
01236                             break;
```

```
01237                         case SegmentType.Line:
01238                             if (i > 0)
01239                             {
01240                                 cachedLength += this.Segments[i].Measure(currPoint);
01241                                 currPoint = this.Segments[i].Point;
01242                             }
01243                             else
01244                             {
01245                                 currPoint = this.Segments[i].Point;
01246                                 figureStartPoint = this.Segments[i].Point;
01247                             }
01248                             break;
01249                         case SegmentType.Arc:
01250                             if (i > 0)
01251                             {
01252                                 cachedLength += this.Segments[i].Measure(currPoint);
01253                                 currPoint = this.Segments[i].Point;
01254                             }
01255                             else
01256                             {
01257                                 ArcSegment seg = (ArcSegment)this.Segments[i];
01258                                 figureStartPoint = new Point(seg.Points[0].X +
    Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01259                                 cachedLength += this.Segments[i].Measure(figureStartPoint);
01260                                 currPoint = this.Segments[i].Point;
01261                             }
01262                             break;
01263                         case SegmentType.Close:
01264                             cachedLength += Math.Sqrt((currPoint.X - figureStartPoint.X) *
    (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
    figureStartPoint.Y));
01265                             currPoint = figureStartPoint;
01266                             break;
01267                         case SegmentType.CubicBezier:
01268                             if (i > 0)
01269                             {
01270                                 cachedLength += this.Segments[i].Measure(currPoint);
01271                                 currPoint = this.Segments[i].Point;
01272                             }
01273                             else
01274                             {
01275                                 currPoint = this.Segments[i].Points[0];
01276                                 figureStartPoint = this.Segments[i].Points[0];
01277                                 cachedLength += this.Segments[i].Measure(currPoint);
01278                                 currPoint = this.Segments[i].Point;
01279                             }
01280                             break;
01281                     }
01282                 }
01283             }
01284 
01285             return cachedLength;
01286         }
01287 
01288 /// <summary>
01289 /// Gets the point at the relative position specified on the <see cref="GraphicsPath"/>.
01290 /// </summary>
01291 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
    1 is the end of the path).</param>
01292 /// <returns>The point at the specified position.</returns>
01293         public Point GetPointAtRelative(double position)
01294         {
01295             return GetPointAtAbsolute(position * this.MeasureLength());
01296         }
01297 
01298 /// <summary>
01299 /// Gets the point at the absolute position specified on the <see cref="GraphicsPath"/>.
01300 /// </summary>
01301 /// <param name="length">The distance to the point from the start of the <see
    cref="GraphicsPath"/>.</param>
01302 /// <returns>The point at the specified position.</returns>
01303         public Point GetPointAtAbsolute(double length)
01304         {
01305             double pathLength = this.MeasureLength();
01306 
01307             if (length >= 0 && length <= pathLength)
01308             {
01309                 double currLen = 0;
01310 
01311                 Point currPoint = new Point();
01312                 Point figureStartPoint = new Point();
01313 
01314                 for (int i = 0; i < this.Segments.Count; i++)
01315                 {
01316                     switch (this.Segments[i].Type)
01317                     {
01318                         case SegmentType.Move:
```

```
01319                               currPoint = this.Segments[i].Point;
01320                               figureStartPoint = this.Segments[i].Point;
01321                               break;
01322                           case SegmentType.Line:
01323                               if (i > 0)
01324                               {
01325                                   double segLength = this.Segments[i].Measure(currPoint);
01326
01327                                   if (currLen + segLength < length)
01328                                   {
01329                                       currLen += segLength;
01330                                       currPoint = this.Segments[i].Point;
01331                                   }
01332                                   else if (segLength > 0)
01333                                   {
01334                                       double pos = (length - currLen) / segLength;
01335                                       return this.Segments[i].GetPointAt(currPoint, pos);
01336                                   }
01337                               }
01338                               else
01339                               {
01340                                   currPoint = this.Segments[i].Point;
01341                                   figureStartPoint = this.Segments[i].Point;
01342                               }
01343                               break;
01344                           case SegmentType.Arc:
01345                               if (i > 0)
01346                               {
01347                                   double segLength = this.Segments[i].Measure(currPoint);
01348
01349                                   if (currLen + segLength < length)
01350                                   {
01351                                       currLen += segLength;
01352                                       currPoint = this.Segments[i].Point;
01353                                   }
01354                                   else if (segLength > 0)
01355                                   {
01356                                       double pos = (length - currLen) / segLength;
01357                                       return this.Segments[i].GetPointAt(currPoint, pos);
01358                                   }
01359                               }
01360                               else
01361                               {
01362                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01363                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01364                                   currPoint = figureStartPoint;
01365
01366                                   double segLength = this.Segments[i].Measure(currPoint);
01367
01368                                   if (currLen + segLength < length)
01369                                   {
01370                                       currLen += segLength;
01371                                       currPoint = this.Segments[i].Point;
01372                                   }
01373                                   else if (segLength > 0)
01374                                   {
01375                                       double pos = (length - currLen) / segLength;
01376                                       return this.Segments[i].GetPointAt(currPoint, pos);
01377                                   }
01378                               }
01379                               break;
01380                           case SegmentType.Close:
01381                               {
01382                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01383
01384                                   if (currLen + segLength < length)
01385                                   {
01386                                       currLen += segLength;
01387                                       currPoint = figureStartPoint;
01388                                   }
01389                                   else
01390                                   {
01391                                       double pos = (length - currLen) / segLength;
01392                                       return new Point(currPoint.X * (1 - pos) + figureStartPoint.X *
      pos, currPoint.Y * (1 - pos) + figureStartPoint.Y * pos);
01393                                   }
01394                               }
01395                               break;
01396                           case SegmentType.CubicBezier:
01397                               if (i > 0)
01398                               {
01399                                   double segLength = this.Segments[i].Measure(currPoint);
01400
01401                                   if (currLen + segLength < length)
```

```
01402                                   {
01403                                       currLen += segLength;
01404                                       currPoint = this.Segments[i].Point;
01405                                   }
01406                                   else if (segLength > 0)
01407                                   {
01408                                       double pos = (length - currLen) / segLength;
01409                                       return this.Segments[i].GetPointAt(currPoint, pos);
01410                                   }
01411                               }
01412                               else
01413                               {
01414                                   currPoint = this.Segments[i].Points[0];
01415                                   figureStartPoint = this.Segments[i].Points[0];
01416                                   double segLength = this.Segments[i].Measure(currPoint);
01417 
01418                                   if (currLen + segLength < length)
01419                                   {
01420                                       currLen += segLength;
01421                                       currPoint = this.Segments[i].Point;
01422                                   }
01423                                   else if (segLength > 0)
01424                                   {
01425                                       double pos = (length - currLen) / segLength;
01426                                       return this.Segments[i].GetPointAt(currPoint, pos);
01427                                   }
01428                               }
01429                               break;
01430                           }
01431                       }
01432 
01433                       throw new InvalidOperationException("Unexpected code path!");
01434                   }
01435                   else if (length > pathLength)
01436                   {
01437                       double currLength = 0;
01438 
01439                       Point currPoint = new Point();
01440                       Point figureStartPoint = new Point();
01441 
01442                       for (int i = 0; i < this.Segments.Count - 1; i++)
01443                       {
01444                           switch (this.Segments[i].Type)
01445                           {
01446                               case SegmentType.Move:
01447                                   currPoint = this.Segments[i].Point;
01448                                   figureStartPoint = this.Segments[i].Point;
01449                                   break;
01450                               case SegmentType.Line:
01451                                   if (i > 0)
01452                                   {
01453                                       currLength += this.Segments[i].Measure(currPoint);
01454                                       currPoint = this.Segments[i].Point;
01455                                   }
01456                                   else
01457                                   {
01458                                       currPoint = this.Segments[i].Point;
01459                                       figureStartPoint = this.Segments[i].Point;
01460                                   }
01461                                   break;
01462                               case SegmentType.Arc:
01463                                   if (i > 0)
01464                                   {
01465                                       currLength += this.Segments[i].Measure(currPoint);
01466                                       currPoint = this.Segments[i].Point;
01467                                   }
01468                                   else
01469                                   {
01470                                       ArcSegment seg = (ArcSegment)this.Segments[i];
01471                                       figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01472                                       currLength += this.Segments[i].Measure(figureStartPoint);
01473                                       currPoint = this.Segments[i].Point;
01474                                   }
01475                                   break;
01476                               case SegmentType.Close:
01477                                   currLength += Math.Sqrt((currPoint.X - figureStartPoint.X) * (currPoint.X
      - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y - figureStartPoint.Y));
01478                                   currPoint = figureStartPoint;
01479                                   break;
01480                               case SegmentType.CubicBezier:
01481                                   if (i > 0)
01482                                   {
01483                                       currLength += this.Segments[i].Measure(currPoint);
01484                                       currPoint = this.Segments[i].Point;
01485                                   }
01486                                   else
```

```
01487                               {
01488                                   currPoint = this.Segments[i].Points[0];
01489                                   figureStartPoint = this.Segments[i].Points[0];
01490                                   currLength += this.Segments[i].Measure(currPoint);
01491                                   currPoint = this.Segments[i].Point;
01492                               }
01493                               break;
01494                       }
01495                   }
01496
01497                   switch (this.Segments[this.Segments.Count - 1].Type)
01498                   {
01499                       case SegmentType.Arc:
01500                       case SegmentType.CubicBezier:
01501                       case SegmentType.Line:
01502                           {
01503                               double pos = 1 + (length - pathLength) / this.Segments[this.Segments.Count
      - 1].Measure(currPoint);
01504                               return this.Segments[this.Segments.Count - 1].GetPointAt(currPoint, pos);
01505                           }
01506                       case SegmentType.Move:
01507                           return currPoint;
01508                       case SegmentType.Close:
01509                           return this.GetPointAtAbsolute(length - pathLength);
01510                   }
01511
01512                   throw new InvalidOperationException("Unexpected code path!");
01513               }
01514               else
01515               {
01516                   Point currPoint = new Point();
01517                   Point figureStartPoint = new Point();
01518
01519                   for (int i = 0; i < this.Segments.Count; i++)
01520                   {
01521                       switch (this.Segments[i].Type)
01522                       {
01523                           case SegmentType.Move:
01524                               currPoint = this.Segments[i].Point;
01525                               figureStartPoint = this.Segments[i].Point;
01526                               break;
01527                           case SegmentType.Line:
01528                               if (i > 0)
01529                               {
01530                                   double segLength = this.Segments[i].Measure(currPoint);
01531                                   double pos = length / segLength;
01532                                   return this.Segments[i].GetPointAt(currPoint, pos);
01533                               }
01534                               else
01535                               {
01536                                   currPoint = this.Segments[i].Point;
01537                                   figureStartPoint = this.Segments[i].Point;
01538                               }
01539                               break;
01540                           case SegmentType.Arc:
01541                               if (i > 0)
01542                               {
01543                                   double segLength = this.Segments[i].Measure(currPoint);
01544                                   double pos = length / segLength;
01545                                   return this.Segments[i].GetPointAt(currPoint, pos);
01546                               }
01547                               else
01548                               {
01549                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01550                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01551                                   currPoint = figureStartPoint;
01552
01553                                   double segLength = this.Segments[i].Measure(currPoint);
01554                                   double pos = length / segLength;
01555                                   return this.Segments[i].GetPointAt(currPoint, pos);
01556                               }
01557                           case SegmentType.Close:
01558                               {
01559                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01560                                   double pos = length / segLength;
01561                                   return new Point(currPoint.X * (1 - pos) + figureStartPoint.X * pos,
      currPoint.Y * (1 - pos) + figureStartPoint.Y * pos);
01562                               }
01563                           case SegmentType.CubicBezier:
01564                               if (i > 0)
01565                               {
01566                                   double segLength = this.Segments[i].Measure(currPoint);
01567                                   double pos = length / segLength;
01568                                   return this.Segments[i].GetPointAt(currPoint, pos);
```

```
01569                               }
01570                               else
01571                               {
01572                                   currPoint = this.Segments[i].Points[0];
01573                                   figureStartPoint = this.Segments[i].Points[0];
01574                                   double segLength = this.Segments[i].Measure(currPoint);
01575                                   double pos = length / segLength;
01576                                   return this.Segments[i].GetPointAt(currPoint, pos);
01577                               }
01578                           }
01579                       }
01580
01581                       throw new InvalidOperationException("Unexpected code path!");
01582                   }
01583               }
01584
01585 /// <summary>
01586 /// Gets the tangent to the point at the relative position specified on the <see
      cref="GraphicsPath"/>.
01587 /// </summary>
01588 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
      1 is the end of the path).</param>
01589 /// <returns>The tangent to the point at the specified position.</returns>
01590         public Point GetTangentAtRelative(double position)
01591         {
01592             return GetTangentAtAbsolute(position * this.MeasureLength());
01593         }
01594
01595 /// <summary>
01596 /// Gets the tangent to the point at the absolute position specified on the <see
      cref="GraphicsPath"/>.
01597 /// </summary>
01598 /// <param name="length">The distance to the point from the start of the <see
      cref="GraphicsPath"/>.</param>
01599 /// <returns>The tangent to the point at the specified position.</returns>
01600         public Point GetTangentAtAbsolute(double length)
01601         {
01602             double pathLength = this.MeasureLength();
01603
01604             if (length >= 0 && length <= pathLength)
01605             {
01606                 double currLen = 0;
01607
01608                 Point currPoint = new Point();
01609                 Point figureStartPoint = new Point();
01610
01611                 for (int i = 0; i < this.Segments.Count; i++)
01612                 {
01613                     switch (this.Segments[i].Type)
01614                     {
01615                         case SegmentType.Move:
01616                             currPoint = this.Segments[i].Point;
01617                             figureStartPoint = this.Segments[i].Point;
01618                             break;
01619                         case SegmentType.Line:
01620                             if (i > 0)
01621                             {
01622                                 double segLength = this.Segments[i].Measure(currPoint);
01623
01624                                 if (currLen + segLength < length)
01625                                 {
01626                                     currLen += segLength;
01627                                     currPoint = this.Segments[i].Point;
01628                                 }
01629                                 else
01630                                 {
01631                                     double pos = (length - currLen) / segLength;
01632                                     return this.Segments[i].GetTangentAt(currPoint, pos);
01633                                 }
01634                             }
01635                             else
01636                             {
01637                                 currPoint = this.Segments[i].Point;
01638                                 figureStartPoint = this.Segments[i].Point;
01639                             }
01640                             break;
01641                         case SegmentType.Arc:
01642                             if (i > 0)
01643                             {
01644                                 double segLength = this.Segments[i].Measure(currPoint);
01645
01646                                 if (currLen + segLength < length)
01647                                 {
01648                                     currLen += segLength;
01649                                     currPoint = this.Segments[i].Point;
01650                                 }
01651                                 else
```

```
01652                                   {
01653                                       double pos = (length - currLen) / segLength;
01654                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01655                                   }
01656                               }
01657                               else
01658                               {
01659                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01660                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01661                                   currPoint = figureStartPoint;
01662
01663                                   double segLength = this.Segments[i].Measure(currPoint);
01664
01665                                   if (currLen + segLength < length)
01666                                   {
01667                                       currLen += segLength;
01668                                       currPoint = this.Segments[i].Point;
01669                                   }
01670                                   else
01671                                   {
01672                                       double pos = (length - currLen) / segLength;
01673                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01674                                   }
01675                               }
01676                               break;
01677                           case SegmentType.Close:
01678                               {
01679                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01680
01681                                   if (currLen + segLength < length)
01682                                   {
01683                                       currLen += segLength;
01684                                       currPoint = figureStartPoint;
01685                                   }
01686                                   else
01687                                   {
01688                                       double pos = (length - currLen) / segLength;
01689                                       return new Point(figureStartPoint.X - currPoint.X,
      figureStartPoint.Y - currPoint.Y).Normalize();
01690                                   }
01691                               }
01692                               break;
01693                           case SegmentType.CubicBezier:
01694                               if (i > 0)
01695                               {
01696                                   double segLength = this.Segments[i].Measure(currPoint);
01697
01698                                   if (currLen + segLength < length)
01699                                   {
01700                                       currLen += segLength;
01701                                       currPoint = this.Segments[i].Point;
01702                                   }
01703                                   else
01704                                   {
01705                                       double pos = (length - currLen) / segLength;
01706                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01707                                   }
01708                               }
01709                               else
01710                               {
01711                                   currPoint = this.Segments[i].Points[0];
01712                                   figureStartPoint = this.Segments[i].Points[0];
01713                                   double segLength = this.Segments[i].Measure(currPoint);
01714
01715                                   if (currLen + segLength < length)
01716                                   {
01717                                       currLen += segLength;
01718                                       currPoint = this.Segments[i].Point;
01719                                   }
01720                                   else
01721                                   {
01722                                       double pos = (length - currLen) / segLength;
01723                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01724                                   }
01725                               }
01726                               break;
01727                       }
01728                   }
01729
01730                   throw new InvalidOperationException("Unexpected code path!");
01731               }
01732               else if (length > pathLength)
01733               {
01734                   double currLength = 0;
```

```
01735
01736                 Point currPoint = new Point();
01737                 Point figureStartPoint = new Point();
01738
01739                 for (int i = 0; i < this.Segments.Count - 1; i++)
01740                 {
01741                     switch (this.Segments[i].Type)
01742                     {
01743                         case SegmentType.Move:
01744                             currPoint = this.Segments[i].Point;
01745                             figureStartPoint = this.Segments[i].Point;
01746                             break;
01747                         case SegmentType.Line:
01748                             if (i > 0)
01749                             {
01750                                 currLength += this.Segments[i].Measure(currPoint);
01751                                 currPoint = this.Segments[i].Point;
01752                             }
01753                             else
01754                             {
01755                                 currPoint = this.Segments[i].Point;
01756                                 figureStartPoint = this.Segments[i].Point;
01757                             }
01758                             break;
01759                         case SegmentType.Arc:
01760                             if (i > 0)
01761                             {
01762                                 currLength += this.Segments[i].Measure(currPoint);
01763                                 currPoint = this.Segments[i].Point;
01764                             }
01765                             else
01766                             {
01767                                 ArcSegment seg = (ArcSegment)this.Segments[i];
01768                                 figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01769                                 currLength += this.Segments[i].Measure(figureStartPoint);
01770                                 currPoint = this.Segments[i].Point;
01771                             }
01772                             break;
01773                         case SegmentType.Close:
01774                             currLength += Math.Sqrt((currPoint.X - figureStartPoint.X) * (currPoint.X
      - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y - figureStartPoint.Y));
01775                             currPoint = figureStartPoint;
01776                             break;
01777                         case SegmentType.CubicBezier:
01778                             if (i > 0)
01779                             {
01780                                 currLength += this.Segments[i].Measure(currPoint);
01781                                 currPoint = this.Segments[i].Point;
01782                             }
01783                             else
01784                             {
01785                                 currPoint = this.Segments[i].Points[0];
01786                                 figureStartPoint = this.Segments[i].Points[0];
01787                                 currLength += this.Segments[i].Measure(currPoint);
01788                                 currPoint = this.Segments[i].Point;
01789                             }
01790                             break;
01791                     }
01792                 }
01793
01794                 switch (this.Segments[this.Segments.Count - 1].Type)
01795                 {
01796                     case SegmentType.Arc:
01797                     case SegmentType.CubicBezier:
01798                     case SegmentType.Line:
01799                         {
01800                             double pos = 1 + (length - pathLength) / this.Segments[this.Segments.Count
      - 1].Measure(currPoint);
01801                             return this.Segments[this.Segments.Count - 1].GetTangentAt(currPoint,
      pos);
01802                         }
01803                     case SegmentType.Move:
01804                         return new Point();
01805                     case SegmentType.Close:
01806                         return this.GetTangentAtAbsolute(length - pathLength);
01807                 }
01808
01809                 throw new InvalidOperationException("Unexpected code path!");
01810             }
01811             else
01812             {
01813                 Point currPoint = new Point();
01814                 Point figureStartPoint = new Point();
01815
01816                 for (int i = 0; i < this.Segments.Count; i++)
01817                 {
```

```
01818                     switch (this.Segments[i].Type)
01819                     {
01820                         case SegmentType.Move:
01821                             currPoint = this.Segments[i].Point;
01822                             figureStartPoint = this.Segments[i].Point;
01823                             break;
01824                         case SegmentType.Line:
01825                             if (i > 0)
01826                             {
01827                                 double segLength = this.Segments[i].Measure(currPoint);
01828                                 double pos = length / segLength;
01829                                 return this.Segments[i].GetTangentAt(currPoint, pos);
01830                             }
01831                             else
01832                             {
01833                                 currPoint = this.Segments[i].Point;
01834                                 figureStartPoint = this.Segments[i].Point;
01835                             }
01836                             break;
01837                         case SegmentType.Arc:
01838                             if (i > 0)
01839                             {
01840                                 double segLength = this.Segments[i].Measure(currPoint);
01841                                 double pos = length / segLength;
01842                                 return this.Segments[i].GetTangentAt(currPoint, pos);
01843                             }
01844                             else
01845                             {
01846                                 ArcSegment seg = (ArcSegment)this.Segments[i];
01847                                 figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01848                                 currPoint = figureStartPoint;
01849 
01850                                 double segLength = this.Segments[i].Measure(currPoint);
01851                                 double pos = length / segLength;
01852                                 return this.Segments[i].GetTangentAt(currPoint, pos);
01853                             }
01854                         case SegmentType.Close:
01855                             {
01856                                 double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01857                                 double pos = length / segLength;
01858                                 return new Point(figureStartPoint.X - currPoint.X, figureStartPoint.Y
      - currPoint.Y).Normalize();
01859                             }
01860                         case SegmentType.CubicBezier:
01861                             if (i > 0)
01862                             {
01863                                 double segLength = this.Segments[i].Measure(currPoint);
01864                                 double pos = length / segLength;
01865                                 return this.Segments[i].GetTangentAt(currPoint, pos);
01866                             }
01867                             else
01868                             {
01869                                 currPoint = this.Segments[i].Points[0];
01870                                 figureStartPoint = this.Segments[i].Points[0];
01871                                 double segLength = this.Segments[i].Measure(currPoint);
01872                                 double pos = length / segLength;
01873                                 return this.Segments[i].GetTangentAt(currPoint, pos);
01874                             }
01875                     }
01876                 }
01877 
01878                 throw new InvalidOperationException("Unexpected code path!");
01879             }
01880         }
01881 
01882 /// <summary>
01883 /// Gets the normal to the point at the absolute position specified on the <see cref="GraphicsPath"/>.
01884 /// </summary>
01885 /// <param name="length">The distance to the point from the start of the <see
      cref="GraphicsPath"/>.</param>
01886 /// <returns>The normal to the point at the specified position.</returns>
01887         public Point GetNormalAtAbsolute(double length)
01888         {
01889             Point tangent = this.GetTangentAtAbsolute(length);
01890             return new Point(-tangent.Y, tangent.X);
01891         }
01892 
01893 /// <summary>
01894 /// Gets the normal to the point at the relative position specified on the <see cref="GraphicsPath"/>.
01895 /// </summary>
01896 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
      1 is the end of the path).</param>
01897 /// <returns>The normal to the point at the specified position.</returns>
01898         public Point GetNormalAtRelative(double position)
```

```
01899         {
01900             Point tangent = this.GetTangentAtRelative(position);
01901             return new Point(-tangent.Y, tangent.X);
01902         }
01903 
01904 /// <summary>
01905 /// Linearises a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments
      that approximate them.
01906 /// </summary>
01907 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
01908 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that approximates the
      current <see cref="GraphicsPath"/>.</returns>
01909         public GraphicsPath Linearise(double resolution)
01910         {
01911             if (!(resolution > 0))
01912             {
01913                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
01914             }
01915 
01916             GraphicsPath tbr = new GraphicsPath();
01917 
01918             Point?[] previousPoints = new Point?[this.Segments.Count];
01919 
01920             previousPoints[0] = null;
01921 
01922             for (int i = 0; i < this.Segments.Count - 1; i++)
01923             {
01924                 if (this.Segments[i].Type != SegmentType.Close)
01925                 {
01926                     previousPoints[i + 1] = this.Segments[i].Point;
01927                 }
01928                 else
01929                 {
01930                     previousPoints[i + 1] = previousPoints[i];
01931                 }
01932             }
01933 
01934             Segment[][] linearisedSegments = new Segment[this.Segments.Count][];
01935 
01936             System.Threading.Tasks.Parallel.For(0, this.Segments.Count, i =>
01937             {
01938                 linearisedSegments[i] = this.Segments[i].Linearise(previousPoints[i],
      resolution).ToArray();
01939             });
01940 
01941             int total = 0;
01942 
01943             for (int i = 0; i < linearisedSegments.Length; i++)
01944             {
01945                 total += linearisedSegments[i].Length;
01946             }
01947 
01948             tbr.Segments.Capacity = total;
01949 
01950             for (int i = 0; i < linearisedSegments.Length; i++)
01951             {
01952                 tbr.Segments.AddRange(linearisedSegments[i]);
01953             }
01954 
01955             return tbr;
01956         }
01957 
01958 /// <summary>
01959 /// Discretises a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments
      that approximate them and ensuring that all line segments are shorter than the specified <paramref
      name="resolution"/>.
01960 /// </summary>
01961 /// <param name="resolution">The maximum length (in absolute units) of line segments in the resulting
      <see cref="GraphicsPath"/>.</param>
01962 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that are shorter than
      <paramref name="resolution"/> and approximate the current <see cref="GraphicsPath"/>.</returns>
01963         public GraphicsPath Discretise(double resolution)
01964         {
01965             if (!(resolution > 0))
01966             {
01967                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
01968             }
01969 
01970             GraphicsPath tbr = new GraphicsPath();
01971 
01972             Point?  previousPoint = null;
01973 
01974             foreach (Segment seg in this.Segments)
01975             {
01976                 tbr.Segments.AddRange(seg.Linearise(previousPoint, resolution));
```

```
01977
01978                 if (seg.Type != SegmentType.Close)
01979                 {
01980                     previousPoint = seg.Point;
01981                 }
01982             }
01983
01984             return Graphics.ReduceMaximumLength(tbr, resolution);
01985         }
01986
01987
01988 /// <summary>
01989 /// Flattens a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path.
01990 /// </summary>
01991 /// <param name="flatness">The maximum deviation from the original path.</param>
01992 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that approximates the
      current <see cref="GraphicsPath"/>.</returns>
01993         public GraphicsPath Flatten(double flatness)
01994         {
01995             if (!(flatness > 0))
01996             {
01997                 throw new ArgumentOutOfRangeException(nameof(flatness), flatness, "The flatness must
      be greater than 0!");
01998             }
01999
02000             GraphicsPath tbr = new GraphicsPath();
02001
02002             Point?[] previousPoints = new Point?[this.Segments.Count];
02003
02004             previousPoints[0] = null;
02005
02006             for (int i = 0; i < this.Segments.Count - 1; i++)
02007             {
02008                 if (this.Segments[i].Type != SegmentType.Close)
02009                 {
02010                     previousPoints[i + 1] = this.Segments[i].Point;
02011                 }
02012                 else
02013                 {
02014                     previousPoints[i + 1] = previousPoints[i];
02015                 }
02016             }
02017
02018             Segment[][] flattenedSegments = new Segment[this.Segments.Count][];
02019
02020             System.Threading.Tasks.Parallel.For(0, this.Segments.Count, i =>
02021             {
02022                 flattenedSegments[i] = this.Segments[i].Flatten(previousPoints[i],
      flatness).ToArray();
02023             });
02024
02025             int total = 0;
02026
02027             for (int i = 0; i < flattenedSegments.Length; i++)
02028             {
02029                 total += flattenedSegments[i].Length;
02030             }
02031
02032             tbr.Segments.Capacity = total;
02033
02034             for (int i = 0; i < flattenedSegments.Length; i++)
02035             {
02036                 tbr.Segments.AddRange(flattenedSegments[i]);
02037             }
02038
02039             return tbr;
02040         }
02041
02042 /// <summary>
02043 /// Gets a collection of the end points of all the segments in the <see cref="GraphicsPath"/>, divided
      by figure.
02044 /// </summary>
02045 /// <returns>A collection of the end points of all the segments in the <see cref="GraphicsPath"/>,
      divided by figure.</returns>
02046         public IEnumerable<List<Point» GetPoints()
02047         {
02048             Point startPoint = new Point();
02049
02050             List<Point> currFigure = null;
02051             bool returned = true;
02052
02053             foreach (Segment seg in this.Segments)
02054             {
02055                 if (seg.Type != SegmentType.Close)
02056                 {
02057                     Point currPoint = seg.Point;
```

```
02058                     if (seg.Type == SegmentType.Move)
02059                     {
02060                         if (!returned)
02061                         {
02062                             yield return currFigure;
02063                         }
02064
02065                         startPoint = currPoint;
02066                         currFigure = new List<Point>();
02067                         returned = false;
02068                     }
02069                     currFigure.Add(currPoint);
02070                 }
02071                 else
02072                 {
02073                     currFigure.Add(startPoint);
02074                     yield return currFigure;
02075                     returned = true;
02076                 }
02077             }
02078
02079             if (!returned)
02080             {
02081                 yield return currFigure;
02082             }
02083         }
02084
02085 /// <summary>
02086 /// Gets a collection of all the figures in the <see cref="GraphicsPath"/>, returned as individual
      <see cref="GraphicsPath"/>s.
02087 /// </summary>
02088 /// <returns>A collection of all the figures in the <see cref="GraphicsPath"/>, returned as individual
      <see cref="GraphicsPath"/>s.</returns>
02089         public IEnumerable<GraphicsPath> GetFigures()
02090         {
02091             GraphicsPath currFigure = null;
02092             bool returned = true;
02093
02094             foreach (Segment seg in this.Segments)
02095             {
02096                 if (seg.Type != SegmentType.Close)
02097                 {
02098                     Point currPoint = seg.Point;
02099                     if (seg.Type == SegmentType.Move)
02100                     {
02101                         if (!returned)
02102                         {
02103                             yield return currFigure;
02104                         }
02105
02106                         currFigure = new GraphicsPath();
02107                         returned = false;
02108                     }
02109
02110                     currFigure.Segments.Add(seg);
02111                 }
02112                 else
02113                 {
02114                     currFigure.Close();
02115                     yield return currFigure;
02116                     returned = true;
02117                 }
02118             }
02119
02120             if (!returned)
02121             {
02122                 yield return currFigure;
02123             }
02124         }
02125
02126
02127 /// <summary>
02128 /// Gets a collection of the tangents at the end point of the segments in which the <see
      cref="GraphicsPath"/> would be linearised, divided by figure.
02129 /// </summary>
02130 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
02131 /// <returns>A collection of the tangents at the end point of the segments in which the <see
      cref="GraphicsPath"/> would be linearised, divided by figure.</returns>
02132         public IEnumerable<List<Point» GetLinearisationPointsNormals(double resolution)
02133         {
02134             if (!(resolution > 0))
02135             {
02136                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
02137             }
02138
02139             Point previousPoint = new Point();
```

```
02140             Point startPoint = new Point();
02141
02142             List<Point> currFigure = null;
02143             bool returned = true;
02144
02145             for (int i = 0; i < this.Segments.Count; i++)
02146             {
02147                 Segment seg = this.Segments[i];
02148
02149                 if (seg.Type != SegmentType.Close)
02150                 {
02151                     Point currPoint = seg.Point;
02152                     if (seg.Type == SegmentType.Move)
02153                     {
02154                         if (!returned)
02155                         {
02156                             yield return currFigure;
02157                         }
02158
02159                         startPoint = currPoint;
02160                         currFigure = new List<Point>();
02161                         returned = false;
02162
02163                         if (i < this.Segments.Count - 1 && this.Segments[i + 1].Type !=
      SegmentType.Move)
02164                         {
02165                             Point tangent = this.Segments[i + 1].GetTangentAt(seg.Point, 0);
02166
02167                             currFigure.Add(new Point(-tangent.Y, tangent.X));
02168                         }
02169                         else
02170                         {
02171                             currFigure.Add(new Point());
02172                         }
02173                     }
02174                     else
02175                     {
02176                         foreach (Point tangent in seg.GetLinearisationTangents(previousPoint,
      resolution))
02177                         {
02178                             currFigure.Add(new Point(-tangent.Y, tangent.X));
02179                         }
02180                     }
02181
02182                     previousPoint = currPoint;
02183                 }
02184                 else
02185                 {
02186                     Point normal;
02187
02188                     if (!startPoint.IsEqual(previousPoint, 1e-4))
02189                     {
02190                         Point tangent = new Point(startPoint.X - previousPoint.X, startPoint.Y -
      previousPoint.Y).Normalize();
02191                         normal = new Point(-tangent.Y, tangent.X);
02192                     }
02193                     else
02194                     {
02195                         normal = currFigure[currFigure.Count - 1];
02196                     }
02197
02198                     currFigure.Add(normal);
02199                     currFigure[0] = new Point((currFigure[1].X + normal.X) * 0.5, (currFigure[1].Y +
      normal.Y) * 0.5).Normalize();
02200
02201                     yield return currFigure;
02202                     returned = true;
02203                 }
02204             }
02205
02206             if (!returned)
02207             {
02208                 yield return currFigure;
02209             }
02210
02211         }
02212
02213
02214         private enum VertexType
02215         {
02216             Start, End, Regular, Split, Merge
02217         };
02218
02219 /// <summary>
02220 /// Divides a <see cref="GraphicsPath"/> into triangles.
02221 /// </summary>
02222 /// <param name="resolution">The resolution that will be used to linearise curve segments in the <see
```

```
cref="GraphicsPath"/>.</param>
/// <param name="clockwise">If this is <see langword="true"/>, the triangles will have their vertices in a clockwise order, otherwise they will be in anticlockwise order.</param>
/// <returns>A collection of distinct <see cref="GraphicsPath"/>s, each representing one triangle.</returns>
        public IEnumerable<GraphicsPath> Triangulate(double resolution, bool clockwise)
        {
            double shiftAmount = 0.01 * resolution;

            if (!(resolution > 0))
            {
                throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution must be greater than 0!");
            }

            GraphicsPath linearisedPath = this.Linearise(resolution);

            List<Point> vertices = new List<Point>();
            List<List<int» vertexEdges = new List<List<int»();
            List<(int, int)> edges = new List<(int, int)>();
            int lastStartingPoint = -1;
            int lastSegmentEnd = -1;
            double area = 0;

            foreach (Segment seg in linearisedPath.Segments)
            {
                if (seg is MoveSegment)
                {
                    vertices.Add(seg.Point);
                    vertexEdges.Add(new List<int>(2));
                    lastStartingPoint = vertices.Count - 1;
                    lastSegmentEnd = vertices.Count - 1;
                }
                else if (seg is LineSegment)
                {
                    if (!vertices[lastSegmentEnd].IsEqual(seg.Point, 1e-4))
                    {
                        vertices.Add(seg.Point);
                        vertexEdges.Add(new List<int>(2));
                        edges.Add((lastSegmentEnd, vertices.Count - 1));
                        area += (seg.Point.X - vertices[lastSegmentEnd].X) * (seg.Point.Y + vertices[lastSegmentEnd].Y);
                        vertexEdges[lastSegmentEnd].Add(edges.Count - 1);
                        vertexEdges[vertices.Count - 1].Add(edges.Count - 1);
                        lastSegmentEnd = vertices.Count - 1;
                    }
                }
                else if (seg is CloseSegment)
                {
                    if (!vertices[lastSegmentEnd].IsEqual(vertices[lastStartingPoint], 1e-4))
                    {
                        edges.Add((lastSegmentEnd, lastStartingPoint));
                        area += (vertices[lastStartingPoint].X - vertices[lastSegmentEnd].X) * (vertices[lastStartingPoint].Y + vertices[lastSegmentEnd].Y);
                        vertexEdges[lastSegmentEnd].Add(edges.Count - 1);
                        vertexEdges[lastStartingPoint].Add(edges.Count - 1);
                    }
                    else
                    {
                        vertices.RemoveAt(lastSegmentEnd);
                        vertexEdges.RemoveAt(lastSegmentEnd);

                        for (int i = 0; i < edges.Count; i++)
                        {
                            if (edges[i].Item1 == lastSegmentEnd)
                            {
                                edges[i] = (lastStartingPoint, edges[i].Item2);
                                vertexEdges[lastStartingPoint].Add(i);
                            }
                            else if (edges[i].Item2 == lastSegmentEnd)
                            {
                                edges[i] = (edges[i].Item1, lastStartingPoint);
                                vertexEdges[lastStartingPoint].Add(i);
                            }
                        }
                    }

                    lastStartingPoint = -1;
                    lastSegmentEnd = -1;
                }
            }

            if (vertices.Count < 3)
            {
                yield break;
            }

```

```
02304             bool isAntiClockwise = area > 0;
02305
02306             int compareVertices(Point a, Point b)
02307             {
02308                 if (a.Y - b.Y != 0)
02309                 {
02310                     return Math.Sign(a.Y - b.Y);
02311                 }
02312                 else
02313                 {
02314                     return Math.Sign(a.X - b.X);
02315                 }
02316             }
02317
02318             Dictionary<double, int> yCoordinates = new Dictionary<double, int>();
02319             Dictionary<double, int> yShiftCount = new Dictionary<double, int>();
02320
02321             foreach (Point pt in vertices)
02322             {
02323                 if (yCoordinates.ContainsKey(pt.Y))
02324                 {
02325                     yCoordinates[pt.Y]++;
02326                 }
02327                 else
02328                 {
02329                     yCoordinates[pt.Y] = 1;
02330                     yShiftCount[pt.Y] = 0;
02331                 }
02332             }
02333
02334             HashSet<double> yS = new HashSet<double>(from el in yCoordinates select el.Key);
02335
02336             for (int i = 0; i < vertices.Count; i++)
02337             {
02338                 if (yCoordinates[vertices[i].Y] > 1)
02339                 {
02340                     int shiftCount = yShiftCount[vertices[i].Y];
02341
02342                     double targetCoordinate;
02343
02344                     do
02345                     {
02346                         shiftCount++;
02347
02348                         targetCoordinate = vertices[i].Y + (2 * (shiftCount % 2) - 1) * (1 -
      Math.Pow(0.5, (shiftCount - 1) / 2 + 1)) * shiftAmount;
02349                     }
02350                     while (yS.Contains(targetCoordinate));
02351
02352                     yS.Add(targetCoordinate);
02353                     yCoordinates[vertices[i].Y]--;
02354                     yShiftCount[vertices[i].Y] = shiftCount;
02355                     vertices[i] = new Point(vertices[i].X, targetCoordinate);
02356                 }
02357
02358             }
02359
02360             Queue<int> sortedVertices = new Queue<int>(Enumerable.Range(0, vertices.Count).OrderBy(i
      => vertices[i], Comparer<Point>.Create(compareVertices)));
02361
02362             VertexType[] vertexTypes = new VertexType[vertices.Count];
02363
02364             List<(int, int)> exploredEdges = new List<(int, int)>();
02365             List<int> helpers = new List<int>();
02366             List<(int, int)> diagonals = new List<(int, int)>();
02367             int[] nexts = new int[vertices.Count];
02368             int[] prevs = new int[vertices.Count];
02369
02370             while (sortedVertices.Count > 0)
02371             {
02372                 int vertex = sortedVertices.Dequeue();
02373
02374                 Point pt = vertices[vertex];
02375
02376                 (int, int) edge1 = edges[vertexEdges[vertex][0]];
02377                 (int, int) edge2 = edges[vertexEdges[vertex][1]];
02378
02379                 int neighbour1 = edge1.Item1 != vertex ?  edge1.Item1 :  edge1.Item2;
02380                 int neighbour2 = edge2.Item1 != vertex ?  edge2.Item1 :  edge2.Item2;
02381
02382                 int minNeighbour = Math.Min(neighbour1, neighbour2);
02383                 int maxNeighbour = Math.Max(neighbour1, neighbour2);
02384
02385                 int prev, next;
02386
02387                 if (vertex - minNeighbour == 1 && maxNeighbour - vertex == 1)
02388                 {
```

```
02389                   prev = minNeighbour;
02390                   next = maxNeighbour;
02391               }
02392               else if ((minNeighbour - vertex == 1 && maxNeighbour - vertex > 1) || vertex -
    maxNeighbour == 1 && vertex - minNeighbour > 1)
02393               {
02394                   prev = maxNeighbour;
02395                   next = minNeighbour;
02396               }
02397               else
02398               {
02399                   throw new InvalidOperationException("Could not make sense of the ordering of the
    vertices!");
02400               }
02401
02402               nexts[vertex] = next;
02403               prevs[vertex] = prev;
02404
02405               Point prevPoint = vertices[prev];
02406               Point nextPoint = vertices[next];
02407
02408               double angle = Math.Atan2(prevPoint.Y - pt.Y, prevPoint.X - pt.X) -
    Math.Atan2(nextPoint.Y - pt.Y, nextPoint.X - pt.X);
02409
02410               if (angle < 0)
02411               {
02412                   angle += 2 * Math.PI;
02413               }
02414
02415               VertexType vertexType;
02416
02417               if (prevPoint.Y >= pt.Y && nextPoint.Y >= pt.Y && angle < Math.PI)
02418               {
02419                   vertexType = VertexType.Start;
02420               }
02421               else if (prevPoint.Y >= pt.Y && nextPoint.Y >= pt.Y && angle > Math.PI)
02422               {
02423                   vertexType = VertexType.Split;
02424               }
02425               else if (prevPoint.Y <= pt.Y && nextPoint.Y <= pt.Y && angle < Math.PI)
02426               {
02427                   vertexType = VertexType.End;
02428               }
02429               else if (prevPoint.Y <= pt.Y && nextPoint.Y <= pt.Y && angle > Math.PI)
02430               {
02431                   vertexType = VertexType.Merge;
02432               }
02433               else
02434               {
02435                   vertexType = VertexType.Regular;
02436               }
02437
02438               vertexTypes[vertex] = vertexType;
02439
02440               //gpr.FillText(vertices[vertex], vertex.ToString(), new Font(new
    FontFamily(FontFamily.StandardFontFamilies.Helvetica), 4), Colours.Orange);
02441
02442               if (vertexType == VertexType.Start)
02443               {
02444                   exploredEdges.Add((prev, vertex));
02445                   helpers.Add(vertex);
02446
02447                   //gpr.StrokeRectangle(pt.X - 1, pt.Y - 1, 2, 2, Colours.Green, 0.25);
02448               }
02449               else if (vertexType == VertexType.End)
02450               {
02451                   int eiM1 = -1;
02452
02453                   for (int i = exploredEdges.Count - 1; i >= 0; i--)
02454                   {
02455                       if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02456                       {
02457                           eiM1 = i;
02458                           break;
02459                       }
02460                   }
02461
02462                   if (eiM1 >= 0)
02463                   {
02464                       if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02465                       {
02466                           diagonals.Add((helpers[eiM1], vertex));
02467                       }
02468
02469                       exploredEdges.RemoveAt(eiM1);
02470                       helpers.RemoveAt(eiM1);
02471                   }
```

```
02472                 }
02473                 else if (vertexType == VertexType.Split)
02474                 {
02475                     (int, int) ej = (-1, -1);
02476                     int ejIndex = -1;
02477 
02478                     double xJ = double.MinValue;
02479 
02480                     for (int i = 0; i < exploredEdges.Count; i++)
02481                     {
02482                         if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02483                         {
02484                             double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02485                             double dx = dy * (vertices[exploredEdges[i].Item2].X -
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02486 
02487                             double x = dx + vertices[exploredEdges[i].Item1].X;
02488 
02489                             if (x < pt.X && x >= xJ)
02490                             {
02491                                 xJ = x;
02492                                 ej = exploredEdges[i];
02493                                 ejIndex = i;
02494                             }
02495                         }
02496                     }
02497 
02498                     if (ejIndex >= 0)
02499                     {
02500                         diagonals.Add((helpers[ejIndex], vertex));
02501 
02502                         helpers[ejIndex] = vertex;
02503 
02504                         exploredEdges.Add((prev, vertex));
02505                         helpers.Add(vertex);
02506                     }
02507                 }
02508                 else if (vertexType == VertexType.Merge)
02509                 {
02510                     int eiM1 = -1;
02511 
02512                     for (int i = exploredEdges.Count - 1; i >= 0; i--)
02513                     {
02514                         if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02515                         {
02516                             eiM1 = i;
02517                             break;
02518                         }
02519                     }
02520 
02521                     if (eiM1 >= 0)
02522                     {
02523                         if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02524                         {
02525                             diagonals.Add((helpers[eiM1], vertex));
02526                         }
02527 
02528                         exploredEdges.RemoveAt(eiM1);
02529                         helpers.RemoveAt(eiM1);
02530                     }
02531 
02532                     (int, int) ej = (-1, -1);
02533                     int ejIndex = -1;
02534 
02535                     double xJ = double.MinValue;
02536 
02537                     for (int i = 0; i < exploredEdges.Count; i++)
02538                     {
02539                         if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02540                         {
02541                             double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02542                             double dx = dy * (vertices[exploredEdges[i].Item2].X -
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02543 
02544                             double x = dx + vertices[exploredEdges[i].Item1].X;
02545 
02546                             if (x < pt.X && x >= xJ)
02547                             {
02548                                 xJ = x;
02549                                 ej = exploredEdges[i];
02550                                 ejIndex = i;
```

```
02551                               }
02552                           }
02553                       }
02554
02555                       if (ejIndex >= 0)
02556                       {
02557                           if (vertexTypes[helpers[ejIndex]] == VertexType.Merge)
02558                           {
02559                               diagonals.Add((helpers[ejIndex], vertex));
02560                           }
02561
02562                           helpers[ejIndex] = vertex;
02563                       }
02564                   }
02565                   else if (vertexType == VertexType.Regular)
02566                   {
02567                       if ((isAntiClockwise && (prevPoint.Y < pt.Y || pt.Y < nextPoint.Y)) ||
      (!isAntiClockwise && (prevPoint.Y > pt.Y || pt.Y > nextPoint.Y)))
02568                       {
02569                           int eiM1 = -1;
02570
02571                           for (int i = exploredEdges.Count - 1; i >= 0; i--)
02572                           {
02573                               if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02574                               {
02575                                   eiM1 = i;
02576                                   break;
02577                               }
02578                           }
02579
02580                           if (eiM1 >= 0)
02581                           {
02582                               if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02583                               {
02584                                   diagonals.Add((helpers[eiM1], vertex));
02585                               }
02586
02587                               exploredEdges.RemoveAt(eiM1);
02588                               helpers.RemoveAt(eiM1);
02589                           }
02590
02591                           exploredEdges.Add((prev, vertex));
02592                           helpers.Add(vertex);
02593                       }
02594                       else
02595                       {
02596                           (int, int) ej = (-1, -1);
02597                           int ejIndex = -1;
02598
02599                           double xJ = double.MinValue;
02600
02601                           for (int i = 0; i < exploredEdges.Count; i++)
02602                           {
02603                               if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02604                               {
02605                                   double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02606                                   double dx = dy * (vertices[exploredEdges[i].Item2].X -
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02607
02608                                   double x = dx + vertices[exploredEdges[i].Item1].X;
02609
02610                                   if (x < pt.X && x >= xJ)
02611                                   {
02612                                       xJ = x;
02613                                       ej = exploredEdges[i];
02614                                       ejIndex = i;
02615                                   }
02616                               }
02617                           }
02618
02619                           if (ejIndex >= 0)
02620                           {
02621                               if (vertexTypes[helpers[ejIndex]] == VertexType.Merge)
02622                               {
02623                                   diagonals.Add((helpers[ejIndex], vertex));
02624                               }
02625
02626                               helpers[ejIndex] = vertex;
02627                           }
02628                       }
02629                   }
02630               }
02631
02632               for (int i = diagonals.Count - 1; i >= 0; i--)
```

```
02633             {
02634                 for (int j = 0; j < edges.Count; j++)
02635                 {
02636                     if (CompareEdges(diagonals[i], edges[j]))
02637                     {
02638                         diagonals.RemoveAt(i);
02639                         break;
02640                     }
02641                 }
02642             }
02643 
02644             List<List<(int, int)» polygons = SplitPolygons(edges, diagonals, vertices, isAntiClockwise
      ?  prevs :  nexts);
02645 
02646             int[] directions = new int[vertices.Count];
02647 
02648             int ind = 0;
02649 
02650             foreach (List<(int, int)> polygon in polygons)
02651             {
02652                 foreach (GraphicsPath pth in TriangulateMonotone(vertices, polygon, directions,
      clockwise ?  -1 :  1))
02653                 {
02654                     yield return pth;
02655                 }
02656 
02657                 ind++;
02658             }
02659         }
02660 
02661         private static bool CompareEdges((int, int) edge1, (int, int) edge2)
02662         {
02663             return (edge1.Item1 == edge2.Item1 && edge1.Item2 == edge2.Item2) || (edge1.Item1 ==
      edge2.Item2 && edge1.Item2 == edge2.Item1);
02664         }
02665 
02666         private static List<List<(int, int)» SplitPolygons(List<(int, int)> edges, List<(int, int)>
      diagonals, List<Point> vertices, int[] nexts)
02667         {
02668             List<List<(int, int)» polygons = new List<List<(int, int)»();
02669 
02670             List<int>[] outPaths = new List<int>[vertices.Count];
02671 
02672             for (int i = 0; i < edges.Count; i++)
02673             {
02674                 if (outPaths[edges[i].Item1] == null)
02675                 {
02676                     outPaths[edges[i].Item1] = new List<int>();
02677                 }
02678 
02679                 if (outPaths[edges[i].Item2] == null)
02680                 {
02681                     outPaths[edges[i].Item2] = new List<int>();
02682                 }
02683 
02684                 outPaths[edges[i].Item1].Add(edges[i].Item2);
02685                 outPaths[edges[i].Item2].Add(edges[i].Item1);
02686             }
02687 
02688             for (int i = 0; i < diagonals.Count; i++)
02689             {
02690                 if (outPaths[diagonals[i].Item1] == null)
02691                 {
02692                     outPaths[diagonals[i].Item1] = new List<int>();
02693                 }
02694 
02695                 if (outPaths[diagonals[i].Item2] == null)
02696                 {
02697                     outPaths[diagonals[i].Item2] = new List<int>();
02698                 }
02699 
02700                 outPaths[diagonals[i].Item1].Add(diagonals[i].Item2);
02701                 outPaths[diagonals[i].Item1].Add(diagonals[i].Item2);
02702                 outPaths[diagonals[i].Item2].Add(diagonals[i].Item1);
02703                 outPaths[diagonals[i].Item2].Add(diagonals[i].Item1);
02704             }
02705 
02706             List<int> activeVertices = (from el in Enumerable.Range(0, outPaths.Length) where
      outPaths[el] != null && outPaths[el].Count > 0 select el).ToList();
02707 
02708             int[] newNexts = new int[nexts.Length];
02709 
02710             for (int i = 0; i < nexts.Length; i++)
02711             {
02712                 if (activeVertices.Contains(i))
02713                 {
02714                     int currNext = nexts[i];
```

```
02715                     while (!outPaths[i].Contains(currNext))
02716                     {
02717                         currNext = nexts[currNext];
02718                     }
02719                     newNexts[i] = currNext;
02720                 }
02721                 else
02722                 {
02723                     newNexts[i] = -1;
02724                 }
02725             }
02726 
02727             while (activeVertices.Count > 0)
02728             {
02729                 List<(int, int)> polygon = new List<(int, int)>();
02730 
02731                 int startPoint = activeVertices.Min();
02732 
02733                 if (!outPaths[startPoint].Contains(newNexts[startPoint]))
02734                 {
02735                     throw new InvalidOperationException("Missing edge!");
02736                 }
02737 
02738                 int prevVertex = startPoint;
02739                 int currVertex = newNexts[startPoint];
02740 
02741                 outPaths[startPoint].Remove(currVertex);
02742                 outPaths[currVertex].Remove(startPoint);
02743                 polygon.Add((startPoint, currVertex));
02744 
02745                 while (currVertex != startPoint)
02746                 {
02747                     Point currPoint = vertices[currVertex];
02748                     Point prevPoint = vertices[prevVertex];
02749 
02750                     double angleIncoming = Math.Atan2(prevPoint.Y - currPoint.Y, prevPoint.X -
      currPoint.X);
02751                     if (angleIncoming < 0)
02752                     {
02753                         angleIncoming += 2 * Math.PI;
02754                     }
02755 
02756                     double maxAngle = double.MinValue;
02757                     int candidateVertex = -1;
02758 
02759                     for (int i = 0; i < outPaths[currVertex].Count; i++)
02760                     {
02761                         double angleI = Math.Atan2(vertices[outPaths[currVertex][i]].Y - currPoint.Y,
      vertices[outPaths[currVertex][i]].X - currPoint.X);
02762                         if (angleI < 0)
02763                         {
02764                             angleI += 2 * Math.PI;
02765                         }
02766                         angleI -= angleIncoming;
02767                         if (angleI < 0)
02768                         {
02769                             angleI += 2 * Math.PI;
02770                         }
02771 
02772                         if (angleI > maxAngle)
02773                         {
02774                             candidateVertex = outPaths[currVertex][i];
02775                             maxAngle = angleI;
02776                         }
02777                     }
02778 
02779                     outPaths[currVertex].Remove(candidateVertex);
02780                     outPaths[candidateVertex].Remove(currVertex);
02781                     polygon.Add((currVertex, candidateVertex));
02782 
02783                     prevVertex = currVertex;
02784                     currVertex = candidateVertex;
02785                 }
02786 
02787                 polygons.Add(polygon);
02788                 activeVertices = (from el in Enumerable.Range(0, outPaths.Length) where outPaths[el]
      != null && outPaths[el].Count > 0 select el).ToList();
02789 
02790                 if (activeVertices.Contains(currVertex))
02791                 {
02792                     int currNext = newNexts[currVertex];
02793                     while (!outPaths[currVertex].Contains(currNext))
02794                     {
02795                         currNext = newNexts[currNext];
02796                     }
02797                     newNexts[currVertex] = currNext;
02798                 }
```

```
02799
02800                 if (activeVertices.Contains(prevVertex))
02801                 {
02802                     int currNext = newNexts[prevVertex];
02803                     while (!outPaths[prevVertex].Contains(currNext))
02804                     {
02805                         currNext = newNexts[currNext];
02806                     }
02807                     newNexts[prevVertex] = currNext;
02808                 }
02809             }
02810
02811             return polygons;
02812         }
02813
02814         private static IEnumerable<GraphicsPath> TriangulateMonotone(List<Point> vertices, List<(int,
      int)> edges, int[] directions, int targetSign)
02815         {
02816             int getDirection((int, int) edge)
02817             {
02818                 if (vertices[edge.Item2].Y != vertices[edge.Item1].Y)
02819                 {
02820                     return Math.Sign(vertices[edge.Item2].Y - vertices[edge.Item1].Y);
02821                 }
02822                 else
02823                 {
02824                     for (int i = 0; i < edges.Count; i++)
02825                     {
02826                         if (edges[i].Item2 == edge.Item1)
02827                         {
02828                             return getDirection(edges[i]);
02829                         }
02830                     }
02831                 }
02832
02833                 throw new InvalidOperationException("Unknown edge direction!");
02834             }
02835
02836
02837             for (int i = 0; i < edges.Count; i++)
02838             {
02839                 directions[edges[i].Item1] = getDirection(edges[i]);
02840             }
02841
02842             int[] sortedVertices = (from el in edges select el.Item1).OrderBy(a => a,
      Comparer<int>.Create((a, b) =>
02843             {
02844                 if (vertices[a].Y != vertices[b].Y)
02845                 {
02846                     return Math.Sign(vertices[a].Y - vertices[b].Y);
02847                 }
02848                 else
02849                 {
02850                     return Math.Sign(vertices[a].X - vertices[b].X);
02851                 }
02852             })).ToArray();
02853
02854             List<(int, int)> diagonals = new List<(int, int)>();
02855
02856             Stack<int> stack = new Stack<int>();
02857
02858             stack.Push(sortedVertices[0]);
02859             stack.Push(sortedVertices[1]);
02860
02861             for (int i = 2; i < sortedVertices.Length - 1; i++)
02862             {
02863                 int onTop = stack.Peek();
02864
02865                 if (directions[sortedVertices[i]] != directions[onTop])
02866                 {
02867                     while (stack.Count > 1)
02868                     {
02869                         int v = stack.Pop();
02870
02871                         diagonals.Add((v, sortedVertices[i]));
02872                     }
02873
02874                     stack.Pop();
02875
02876                     stack.Push(sortedVertices[i - 1]);
02877                     stack.Push(sortedVertices[i]);
02878                 }
02879                 else
02880                 {
02881                     int lastPopped = stack.Pop();
02882
02883                     bool shouldContinue = true;
```

```
02884
02885                     while (shouldContinue && stack.Count > 0)
02886                     {
02887                         int currVert = stack.Peek();
02888
02889                         double areaSign = Math.Sign((vertices[currVert].X - vertices[lastPopped].X) *
      (vertices[sortedVertices[i]].Y - vertices[lastPopped].Y) - (vertices[currVert].Y -
      vertices[lastPopped].Y) * (vertices[sortedVertices[i]].X - vertices[lastPopped].X));
02890
02891                         double dirSign = Math.Sign(vertices[currVert].X - vertices[lastPopped].X);
02892
02893                         if (areaSign * dirSign > 0)
02894                         {
02895                             lastPopped = stack.Pop();
02896
02897                             diagonals.Add((currVert, sortedVertices[i]));
02898                         }
02899                         else
02900                         {
02901                             shouldContinue = false;
02902                         }
02903                     }
02904
02905                     stack.Push(lastPopped);
02906                     stack.Push(sortedVertices[i]);
02907                 }
02908             }
02909
02910             stack.Pop();
02911
02912             while (stack.Count > 1)
02913             {
02914                 int v = stack.Pop();
02915
02916                 diagonals.Add((v, sortedVertices[sortedVertices.Length - 1]));
02917             }
02918
02919             List<int>[] connections = new List<int>[vertices.Count];
02920
02921             for (int i = 0; i < edges.Count; i++)
02922             {
02923                 connections[edges[i].Item1] = new List<int>();
02924             }
02925
02926             for (int i = 0; i < edges.Count; i++)
02927             {
02928                 connections[edges[i].Item1].Add(edges[i].Item2);
02929                 connections[edges[i].Item2].Add(edges[i].Item1);
02930             }
02931
02932             for (int i = 0; i < diagonals.Count; i++)
02933             {
02934                 connections[diagonals[i].Item1].Add(diagonals[i].Item2);
02935                 connections[diagonals[i].Item1].Add(diagonals[i].Item2);
02936
02937                 connections[diagonals[i].Item2].Add(diagonals[i].Item1);
02938                 connections[diagonals[i].Item2].Add(diagonals[i].Item1);
02939             }
02940
02941             int totalTriangles = (edges.Count + diagonals.Count * 2) / 3;
02942
02943             List<List<(int, int)» polygons = new List<List<(int, int)»();
02944
02945             while (polygons.Count < totalTriangles)
02946             {
02947                 int p1 = -1;
02948                 int p2 = -1;
02949
02950                 for (int i = 0; i < connections.Length; i++)
02951                 {
02952                     if (connections[i] != null && connections[i].Count > 0)
02953                     {
02954                         p1 = i;
02955                         p2 = connections[i][0];
02956
02957                         connections[i].Remove(p2);
02958                         connections[p2].Remove(i);
02959                         break;
02960                     }
02961                 }
02962
02963                 int p3 = -1;
02964
02965                 for (int i = 0; i < connections[p1].Count; i++)
02966                 {
02967                     if (connections[connections[p1][i]].Contains(p2))
02968                     {
```

```
02969                           p3 = connections[p1][i];
02970                           connections[p1].Remove(p3);
02971                           connections[p2].Remove(p3);
02972                           connections[p3].Remove(p1);
02973                           connections[p3].Remove(p2);
02974                           break;
02975                       }
02976                   }
02977
02978                   int sign = Math.Sign((vertices[p1].X - vertices[p2].X) * (vertices[p3].Y -
      vertices[p2].Y) - (vertices[p1].Y - vertices[p2].Y) * (vertices[p3].X - vertices[p2].X));
02979
02980                   if (sign == targetSign)
02981                   {
02982                       polygons.Add(new List<(int, int)>() { (p1, p2), (p2, p3), (p3, p1) });
02983                   }
02984                   else
02985                   {
02986                       polygons.Add(new List<(int, int)>() { (p1, p3), (p3, p2), (p2, p1) });
02987                   }
02988               }
02989
02990               foreach (List<(int, int)> polygon in polygons)
02991               {
02992                   GraphicsPath polygonPath = new GraphicsPath();
02993                   foreach ((int, int) edge in polygon)
02994                   {
02995                       if (polygonPath.Segments.Count == 0)
02996                       {
02997                           polygonPath.MoveTo(vertices[edge.Item1]);
02998                       }
02999
03000                       polygonPath.LineTo(vertices[edge.Item2]);
03001                   }
03002
03003                   yield return polygonPath;
03004               }
03005           }
03006
03007 /// <summary>
03008 /// Transforms all of the <see cref="Point"/>s in the <see cref="GraphicsPath"/> with an arbitrary
      transformation function.
03009 /// </summary>
03010 /// <param name="transformationFunction">An arbitrary transformation function.</param>
03011 /// <returns>A new <see cref="GraphicsPath"/> in which all points have been replaced using the
      <paramref name="transformationFunction"/>.</returns>
03012           public GraphicsPath Transform(Func<Point, Point> transformationFunction)
03013           {
03014               GraphicsPath tbr = new GraphicsPath();
03015
03016               foreach (Segment seg in this.Segments)
03017               {
03018                   tbr.Segments.AddRange(seg.Transform(transformationFunction));
03019               }
03020
03021               return tbr;
03022           }
03023
03024
03025           private Rectangle cachedBounds = Rectangle.NaN;
03026
03027 /// <summary>
03028 /// Compute the rectangular bounds of the path.
03029 /// </summary>
03030 /// <returns>The smallest <see cref="Rectangle"/> that contains the path.</returns>
03031           public Rectangle GetBounds()
03032           {
03033               if (double.IsNaN(cachedBounds.Location.X) || double.IsNaN(cachedBounds.Location.Y) ||
      double.IsNaN(cachedBounds.Size.Width) || double.IsNaN(cachedBounds.Size.Height))
03034               {
03035                   Point min = new Point(double.MaxValue, double.MaxValue);
03036                   Point max = new Point(double.MinValue, double.MinValue);
03037
03038                   Point currPoint = new Point();
03039                   Point figureStartPoint = new Point();
03040
03041                   for (int i = 0; i < this.Segments.Count; i++)
03042                   {
03043                       switch (this.Segments[i].Type)
03044                       {
03045                           case SegmentType.Move:
03046                               currPoint = this.Segments[i].Point;
03047                               figureStartPoint = this.Segments[i].Point;
03048                               min = Point.Min(min, this.Segments[i].Point);
03049                               max = Point.Max(max, this.Segments[i].Point);
03050                               break;
03051                           case SegmentType.Line:
```

```
03052                           if (i > 0)
03053                           {
03054                               currPoint = this.Segments[i].Point;
03055                           }
03056                           else
03057                           {
03058                               currPoint = this.Segments[i].Point;
03059                               figureStartPoint = this.Segments[i].Point;
03060                           }
03061                           min = Point.Min(min, this.Segments[i].Point);
03062                           max = Point.Max(max, this.Segments[i].Point);
03063                           break;
03064                       case SegmentType.Arc:
03065                           ArcSegment seg = (ArcSegment)this.Segments[i];
03066
03067                           if (i == 0)
03068                           {
03069                               figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
03070                               currPoint = figureStartPoint;
03071
03072                               min = Point.Min(min, currPoint);
03073                               max = Point.Max(max, currPoint);
03074                           }
03075
03076                           double theta1 = seg.StartAngle;
03077                           double theta2 = seg.EndAngle;
03078
03079                           if (theta1 > theta2)
03080                           {
03081                               (theta2, theta1) = (theta1, theta2);
03082                           }
03083
03084                           while (theta1 < 0)
03085                           {
03086                               theta1 += 2 * Math.PI;
03087                               theta2 += 2 * Math.PI;
03088                           }
03089
03090                           theta1 /= 2 * Math.PI;
03091                           theta2 /= 2 * Math.PI;
03092
03093                           int minAngle = (int)Math.Min(0, Math.Floor(theta1)) * 4;
03094                           int maxAngle = (int)Math.Max(1, Math.Ceiling(theta2)) * 4;
03095
03096                           theta1 *= 4;
03097                           theta2 *= 4;
03098
03099                           bool right = false;
03100                           bool bottom = false;
03101                           bool left = false;
03102                           bool top = false;
03103
03104                           for (int j = minAngle; j < maxAngle; j++)
03105                           {
03106                               if (theta1 <= j && theta2 >= j)
03107                               {
03108                                   switch (j % 4)
03109                                   {
03110                                       case 0:
03111                                           right = true;
03112                                           break;
03113                                       case 1:
03114                                           bottom = true;
03115                                           break;
03116                                       case 2:
03117                                           left = true;
03118                                           break;
03119                                       case 3:
03120                                           top = true;
03121                                           break;
03122                                   }
03123                               }
03124                           }
03125
03126                           if (right)
03127                           {
03128                               Point p = new Point(seg.Points[0].X + seg.Radius, seg.Points[0].Y);
03129                               min = Point.Min(min, p);
03130                               max = Point.Max(max, p);
03131                           }
03132
03133                           if (bottom)
03134                           {
03135                               Point p = new Point(seg.Points[0].X, seg.Points[0].Y + seg.Radius);
03136                               min = Point.Min(min, p);
03137                               max = Point.Max(max, p);
```

```
03138                               }
03139
03140                               if (left)
03141                               {
03142                                   Point p = new Point(seg.Points[0].X - seg.Radius, seg.Points[0].Y);
03143                                   min = Point.Min(min, p);
03144                                   max = Point.Max(max, p);
03145                               }
03146
03147                               if (top)
03148                               {
03149                                   Point p = new Point(seg.Points[0].X, seg.Points[0].Y - seg.Radius);
03150                                   min = Point.Min(min, p);
03151                                   max = Point.Max(max, p);
03152                               }
03153
03154                               min = Point.Min(min, this.Segments[i].Point);
03155                               max = Point.Max(max, this.Segments[i].Point);
03156
03157                               currPoint = this.Segments[i].Point;
03158                               break;
03159                           case SegmentType.Close:
03160                               currPoint = figureStartPoint;
03161                               break;
03162                           case SegmentType.CubicBezier:
03163                               if (i == 0)
03164                               {
03165                                   currPoint = this.Segments[i].Points[0];
03166                                   figureStartPoint = this.Segments[i].Points[0];
03167                               }
03168
03169                               min = Point.Min(min, this.Segments[i].Point);
03170                               max = Point.Max(max, this.Segments[i].Point);
03171
03172                               {
03173                                   double aX = -currPoint.X + 3 * this.Segments[i].Points[0].X - 3 *
      this.Segments[i].Points[1].X + this.Segments[i].Point.X;
03174                                   double bX = 2 * (currPoint.X - 2 * this.Segments[i].Points[0].X +
      this.Segments[i].Points[1].X);
03175                                   double cX = this.Segments[i].Points[0].X - currPoint.X;
03176
03177                                   double t1X;
03178                                   double t2X;
03179
03180                                   if (Math.Abs(aX) < 1e-5)
03181                                   {
03182                                       if (Math.Abs(bX) >= 1e-5)
03183                                       {
03184                                           t1X = -cX / bX;
03185                                           t2X = t1X;
03186                                       }
03187                                       else
03188                                       {
03189                                           t1X = double.NaN;
03190                                           t2X = double.NaN;
03191                                       }
03192                                   }
03193                                   else
03194                                   {
03195                                       double delta = bX * bX - 4 * aX * cX;
03196
03197                                       if (delta >= -1e-5)
03198                                       {
03199                                           delta = Math.Max(delta, 0);
03200
03201                                           delta = Math.Sqrt(delta);
03202                                           t1X = (-bX + delta) / (2 * aX);
03203                                           t2X = (-bX - delta) / (2 * aX);
03204                                       }
03205                                       else
03206                                       {
03207                                           t1X = double.NaN;
03208                                           t2X = double.NaN;
03209                                       }
03210                                   }
03211
03212                                   if (t1X >= 0 && t1X <= 1)
03213                                   {
03214                                       Point p1 =
      ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t1X);
03215                                       min = Point.Min(min, p1);
03216                                       max = Point.Max(max, p1);
03217                                   }
03218
03219                                   if (t2X >= 0 && t2X <= 1)
03220                                   {
03221                                       Point p2 =
```

```
    ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t2X);
03222                                   min = Point.Min(min, p2);
03223                                   max = Point.Max(max, p2);
03224                               }
03225                           }
03226
03227                           {
03228                               double aY = -currPoint.Y + 3 * this.Segments[i].Points[0].Y - 3 *
    this.Segments[i].Points[1].Y + this.Segments[i].Point.Y;
03229                               double bY = 2 * (currPoint.Y - 2 * this.Segments[i].Points[0].Y +
    this.Segments[i].Points[1].Y);
03230                               double cY = this.Segments[i].Points[0].Y - currPoint.Y;
03231
03232                               double t1Y;
03233                               double t2Y;
03234
03235                               if (Math.Abs(aY) < 1e-5)
03236                               {
03237                                   if (Math.Abs(bY) >= 1e-5)
03238                                   {
03239                                       t1Y = -cY / bY;
03240                                       t2Y = t1Y;
03241                                   }
03242                                   else
03243                                   {
03244                                       t1Y = double.NaN;
03245                                       t2Y = double.NaN;
03246                                   }
03247                               }
03248                               else
03249                               {
03250                                   double delta = bY * bY - 4 * aY * cY;
03251
03252                                   if (delta >= -1e-5)
03253                                   {
03254                                       delta = Math.Max(delta, 0);
03255
03256                                       delta = Math.Sqrt(delta);
03257                                       t1Y = (-bY + delta) / (2 * aY);
03258                                       t2Y = (-bY - delta) / (2 * aY);
03259                                   }
03260                                   else
03261                                   {
03262                                       t1Y = double.NaN;
03263                                       t2Y = double.NaN;
03264                                   }
03265                               }
03266
03267                               if (t1Y >= 0 && t1Y <= 1)
03268                               {
03269                                   Point p1 =
    ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t1Y);
03270                                   min = Point.Min(min, p1);
03271                                   max = Point.Max(max, p1);
03272                               }
03273
03274                               if (t2Y >= 0 && t2Y <= 1)
03275                               {
03276                                   Point p2 =
    ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t2Y);
03277                                   min = Point.Min(min, p2);
03278                                   max = Point.Max(max, p2);
03279                               }
03280                           }
03281
03282                           currPoint = this.Segments[i].Point;
03283                           break;
03284                       }
03285                   }
03286
03287                   cachedBounds = new Rectangle(min, max);
03288               }
03289
03290               return cachedBounds;
03291           }
03292
03293           internal GraphicsPath ConvertArcsToBeziers()
03294           {
03295               GraphicsPath tbr = new GraphicsPath();
03296
03297               foreach (Segment seg in this.Segments)
03298               {
03299                   if (seg is ArcSegment arc)
03300                   {
03301                       tbr.Segments.AddRange(arc.ToBezierSegments());
03302                   }
03303                   else
```

```
03304                 {
03305                     tbr.Segments.Add(seg);
03306                 }
03307             }
03308 
03309             return tbr;
03310         }
03311 
03312         private static Point IntersectLines(Point p1, Point d1, Point p3, Point d3)
03313         {
03314             Point p2 = p1 + d1;
03315             Point p4 = p3 + d3;
03316             double denom = d1.X * d3.Y - d1.Y * d3.X;
03317 
03318             double x = (-(p1.X * p2.Y - p1.Y * p2.X) * d3.X + d1.X * (p3.X * p4.Y - p3.Y * p4.X)) /
      denom;
03319             double y = (-(p1.X * p2.Y - p1.Y * p2.X) * d3.Y + d1.Y * (p3.X * p4.Y - p3.Y * p4.X)) /
      denom;
03320 
03321             return new Point(x, y);
03322         }
03323 
03324 /// <summary>
03325 /// Reverses the <see cref="GraphicsPath"/>.
03326 /// </summary>
03327 /// <returns>The reversed <see cref="GraphicsPath"/>.</returns>
03328         public GraphicsPath Reverse()
03329         {
03330             GraphicsPath tbr = new GraphicsPath();
03331 
03332             foreach (GraphicsPath figure in this.GetFigures())
03333             {
03334                 Point currPoint = new Point();
03335                 Point startPoint = new Point();
03336                 bool started = false;
03337 
03338                 List<(int type, Point p1, Point p2, Point p3, Point p4)> invertedSegments = new
      List<(int type, Point p1, Point p2, Point p3, Point p4)>();
03339 
03340                 for (int i = 0; i < figure.Segments.Count; i++)
03341                 {
03342                     if (figure.Segments[i].Type == SegmentType.Move)
03343                     {
03344                         currPoint = figure.Segments[i].Point;
03345                         startPoint = currPoint;
03346                         started = true;
03347                     }
03348                     else if (figure.Segments[i].Type == SegmentType.Line)
03349                     {
03350                         invertedSegments.Add((0, figure.Segments[i].Point, currPoint, new Point(), new
      Point()));
03351                         currPoint = figure.Segments[i].Point;
03352 
03353                         if (!started)
03354                         {
03355                             started = true;
03356                             startPoint = currPoint;
03357                         }
03358                     }
03359                     else if (figure.Segments[i].Type == SegmentType.Arc && figure.Segments[i] is
      ArcSegment arc)
03360                     {
03361                         invertedSegments.Add((1, figure.Segments[i].Points[0], new Point(arc.Radius,
      0), new Point(arc.EndAngle, arc.StartAngle), new Point()));
03362                         currPoint = figure.Segments[i].GetPointAt(currPoint, 1);
03363 
03364                         if (!started)
03365                         {
03366                             started = true;
03367                             startPoint = figure.Segments[i].GetPointAt(currPoint, 0);
03368                         }
03369                     }
03370                     else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03371                     {
03372                         invertedSegments.Add((2, figure.Segments[i].Points[2],
      figure.Segments[i].Points[1], figure.Segments[i].Points[0], currPoint));
03373                         currPoint = figure.Segments[i].GetPointAt(currPoint, 1);
03374 
03375                         if (!started)
03376                         {
03377                             started = true;
03378                             startPoint = figure.Segments[i].GetPointAt(currPoint, 0);
03379                         }
03380                     }
03381                     else if (figure.Segments[i].Type == SegmentType.Close)
03382                     {
03383                         invertedSegments.Add((3, startPoint, currPoint, new Point(), new Point()));
```

```
03384                   }
03385               }
03386
03387               bool isClosed = false;
03388
03389               for (int i = invertedSegments.Count - 1; i >= 0; i--)
03390               {
03391                   if (invertedSegments[i].type == 3)
03392                   {
03393                       tbr.MoveTo(invertedSegments[i].p1);
03394                       tbr.LineTo(invertedSegments[i].p2);
03395                       isClosed = true;
03396                   }
03397                   else if (invertedSegments[i].type == 0)
03398                   {
03399                       if (i == invertedSegments.Count - 1)
03400                       {
03401                           tbr.MoveTo(invertedSegments[i].p1);
03402                       }
03403
03404                       tbr.LineTo(invertedSegments[i].p2);
03405                   }
03406                   else if (invertedSegments[i].type == 1)
03407                   {
03408                       tbr.Arc(invertedSegments[i].p1, invertedSegments[i].p2.X,
    invertedSegments[i].p3.X, invertedSegments[i].p3.Y);
03409                   }
03410                   else if (invertedSegments[i].type == 2)
03411                   {
03412                       if (i == invertedSegments.Count - 1)
03413                       {
03414                           tbr.MoveTo(invertedSegments[i].p1);
03415                       }
03416
03417                       tbr.CubicBezierTo(invertedSegments[i].p2, invertedSegments[i].p3,
    invertedSegments[i].p4);
03418                   }
03419               }
03420
03421               if (isClosed)
03422               {
03423                   tbr.Close();
03424               }
03425           }
03426
03427           return tbr;
03428       }
03429
03430 /// <summary>
03431 /// Returns a <see cref="GraphicsPath"/> representing the stroke of the current <see
    cref="GraphicsPath"/>.
03432 /// </summary>
03433 /// <param name="lineWidth">The thickness of the stroke.</param>
03434 /// <param name="lineCap">The line cap used in the stroke.</param>
03435 /// <param name="lineJoin">The line join used in the stroke.</param>
03436 /// <returns>A <see cref="GraphicsPath"/> representing the stroke of the current <see
    cref="GraphicsPath"/>.</returns>
03437       public GraphicsPath GetStroke(double lineWidth = 1, LineCaps lineCap = LineCaps.Butt,
    LineJoins lineJoin = LineJoins.Miter)
03438       {
03439           double flatness = lineWidth * 0.0001;
03440
03441           GraphicsPath tbr = new GraphicsPath();
03442
03443           foreach (GraphicsPath figure in this.GetFigures())
03444           {
03445               GraphicsPath half1 = new GraphicsPath();
03446
03447               ProcessHalfStroke(figure, half1, lineWidth, lineJoin, flatness, true);
03448
03449               tbr.AddPath(half1);
03450
03451               GraphicsPath half2 = new GraphicsPath();
03452               ProcessHalfStroke(figure, half2, lineWidth, lineJoin, flatness, false);
03453
03454               half2 = half2.Reverse();
03455
03456               if (figure.Segments[figure.Segments.Count - 1].Type != SegmentType.Close &&
    half1.Segments.Count > 0 && half2.Segments.Count > 0)
03457               {
03458                   if (lineCap == LineCaps.Butt)
03459                   {
03460                       if (half2.Segments[0].Type == SegmentType.Move)
03461                       {
03462                           half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03463                       }
03464
```

```
03465                       tbr.AddPath(half2);
03466                       tbr.Close();
03467                   }
03468                   else if (lineCap == LineCaps.Square)
03469                   {
03470                       if (half2.Segments[0].Type == SegmentType.Move)
03471                       {
03472                           half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03473                       }
03474
03475                       Point cap1 = tbr.GetPointAtRelative(1);
03476                       Point cap2 = half2.GetPointAtRelative(0);
03477
03478                       Point tangent = this.GetTangentAtRelative(1);
03479
03480                       tbr.LineTo(cap1 + tangent * lineWidth * 0.5);
03481                       tbr.LineTo(cap2 + tangent * lineWidth * 0.5);
03482
03483                       tbr.AddPath(half2);
03484
03485                       Point cap3 = tbr.GetPointAtRelative(0);
03486                       Point cap4 = half2.GetPointAtRelative(1);
03487
03488                       Point tangent2 = this.GetTangentAtRelative(0) * -1;
03489
03490                       tbr.LineTo(cap4 + tangent2 * lineWidth * 0.5);
03491                       tbr.LineTo(cap3 + tangent2 * lineWidth * 0.5);
03492
03493                       tbr.Close();
03494                   }
03495                   else if (lineCap == LineCaps.Round)
03496                   {
03497                       if (half2.Segments[0].Type == SegmentType.Move)
03498                       {
03499                           half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03500                       }
03501
03502                       Point cap1 = tbr.GetPointAtRelative(1);
03503                       Point cap2 = half2.GetPointAtRelative(0);
03504
03505                       Point center = this.GetPointAtRelative(1);
03506
03507                       double ang1 = (Math.Atan2(cap1.Y - center.Y, cap1.X - center.X) + 2 * Math.PI)
      % (2 * Math.PI);
03508                       double ang2 = (Math.Atan2(cap2.Y - center.Y, cap2.X - center.X) + 2 * Math.PI)
      % (2 * Math.PI);
03509
03510                       tbr.Arc(center, lineWidth * 0.5, ang1, ang2);
03511
03512
03513                       tbr.AddPath(half2);
03514
03515                       Point cap3 = tbr.GetPointAtRelative(0);
03516                       Point cap4 = half2.GetPointAtRelative(1);
03517
03518                       Point center2 = this.GetPointAtRelative(0);
03519
03520                       double ang3 = (Math.Atan2(cap4.Y - center2.Y, cap4.X - center2.X) + 2 *
      Math.PI) % (2 * Math.PI);
03521                       double ang4 = (Math.Atan2(cap3.Y - center2.Y, cap3.X - center2.X) + 2 *
      Math.PI) % (2 * Math.PI);
03522
03523                       tbr.Arc(center2, lineWidth * 0.5, ang3, ang4);
03524
03525                       tbr.Close();
03526                   }
03527               }
03528               else
03529               {
03530                   tbr.AddPath(half2);
03531               }
03532           }
03533
03534           return RemoveDuplicates(tbr);
03535       }
03536
03537       // Adapted from https://www.particleincell.com/2013/cubic-line-intersection/
03538       private static double[] CubicRoots(double a, double b, double c, double d)
03539       {
03540           double A = b / a;
03541           double B = c / a;
03542           double C = d / a;
03543
03544           double Q = (3 * B - A * A) / 9;
03545           double R = (9 * A * B - 27 * C - 2 * A * A * A) / 54;
03546           double D = Q * Q * Q + R * R; // polynomial discriminant
03547
```

```
03548             if (Math.Abs(D) < Tolerance)
03549             {
03550                 D = 0;
03551             }
03552 
03553             double[] t = new double[3];
03554 
03555             if (D >= 0)                                 // complex or duplicate roots
03556             {
03557                 var S = Math.Sign(R + Math.Sqrt(D)) * Math.Pow(Math.Abs(R + Math.Sqrt(D)), (1.0 / 3));
03558                 var T = Math.Sign(R - Math.Sqrt(D)) * Math.Pow(Math.Abs(R - Math.Sqrt(D)), (1.0 / 3));
03559 
03560                 t[0] = -A / 3 + (S + T);                    // real root
03561                 t[1] = -A / 3 - (S + T) / 2;                  // real part of complex root
03562                 t[2] = -A / 3 - (S + T) / 2;                  // real part of complex root
03563                 double Im = Math.Abs(Math.Sqrt(3) * (S - T) / 2);    // complex part of root pair
03564 
03565                 /*discard complex roots*/
03566                 if (Im > Tolerance)
03567                 {
03568                     t[1] = -1;
03569                     t[2] = -1;
03570                 }
03571 
03572             }
03573             else                                          // distinct real roots
03574             {
03575                 var th = Math.Acos(R / Math.Sqrt(-Math.Pow(Q, 3)));
03576 
03577                 t[0] = 2 * Math.Sqrt(-Q) * Math.Cos(th / 3) - A / 3;
03578                 t[1] = 2 * Math.Sqrt(-Q) * Math.Cos((th + 2 * Math.PI) / 3) - A / 3;
03579                 t[2] = 2 * Math.Sqrt(-Q) * Math.Cos((th + 4 * Math.PI) / 3) - A / 3;
03580             }
03581 
03582             return new HashSet<double>(t.Select(x => Math.Abs(x) <= Tolerance ?  0 :  Math.Abs(x - 1)
    <= Tolerance ?  1 :  x).Where(x => x >= 0 && x <= 1)).ToArray();
03583         }
03584 
03585 /// <summary>
03586 /// Determines whether the specified <paramref name="point"/> falls within the current <see
    cref="GraphicsPath"/>.
03587 /// </summary>
03588 /// <param name="point">The <see cref="Point"/> being analysed.</param>
03589 /// <param name="fillRule">The rule to use to determine whether a point is inside or outside of a the
    <see cref="GraphicsPath"/>.</param>
03590 /// <returns><see langword="true"/> if the <paramref name="point"/> is inside the <see
    cref="GraphicsPath"/>, or <see langword="false"/> if it is outside.  If the point lies exactly on the
    <see cref="GraphicsPath"/> (or very close to it), the result may be either <see langword="true"/> or
    <see langword="false"/>, depending on numerical approximations.</returns>
03591         public bool ContainsPoint(Point point, FillRule fillRule)
03592         {
03593             return ContainsPointInternal(point, fillRule == FillRule.EvenOdd) > 0;
03594         }
03595 
03596         private GraphicsPath MonotoniseOnY()
03597         {
03598             GraphicsPath tbr = new GraphicsPath();
03599             Point startPoint = new Point();
03600             Point currPoint = new Point();
03601 
03602 
03603             foreach (Segment seg in this.Segments)
03604             {
03605                 switch (seg.Type)
03606                 {
03607                     case SegmentType.Move:
03608                         tbr.Segments.Add(seg);
03609                         currPoint = seg.Point;
03610                         startPoint = seg.Point;
03611                         break;
03612 
03613                     case SegmentType.Line:
03614                         tbr.Segments.Add(seg);
03615                         currPoint = seg.Point;
03616                         break;
03617 
03618                     case SegmentType.Close:
03619                         tbr.Segments.Add(seg);
03620                         currPoint = startPoint;
03621                         break;
03622 
03623                     case SegmentType.Arc:
03624                         foreach (Segment seg2 in ((ArcSegment)seg).ToBezierSegments())
03625                         {
03626                             if (seg2 is CubicBezierSegment cub)
03627                             {
03628                                 tbr.Segments.AddRange(cub.MonotoniseOnY(currPoint));
```

```
03629                                   currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03630                               }
03631                               else
03632                               {
03633                                   tbr.Segments.Add(seg2);
03634                                   currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03635                               }
03636                           }
03637                           break;
03638                       case SegmentType.CubicBezier:
03639                           tbr.Segments.AddRange(((CubicBezierSegment)seg).MonotoniseOnY(currPoint));
03640                           currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03641                           break;
03642                   }
03643               }
03644
03645               return tbr;
03646           }
03647
03648           private GraphicsPath Monotonise(double flatness)
03649           {
03650               GraphicsPath tbr = new GraphicsPath();
03651               Point startPoint = new Point();
03652               Point currPoint = new Point();
03653
03654               foreach (Segment seg in this.Segments)
03655               {
03656                   switch (seg.Type)
03657                   {
03658                       case SegmentType.Move:
03659                           tbr.Segments.Add(seg);
03660                           currPoint = seg.Point;
03661                           startPoint = seg.Point;
03662                           break;
03663
03664                       case SegmentType.Line:
03665                           tbr.Segments.Add(seg);
03666                           currPoint = seg.Point;
03667                           break;
03668
03669                       case SegmentType.Close:
03670                           tbr.Segments.Add(seg);
03671                           currPoint = startPoint;
03672                           break;
03673
03674                       case SegmentType.Arc:
03675                           foreach (Segment seg2 in ((ArcSegment)seg).ToBezierSegments())
03676                           {
03677                               if (seg2 is CubicBezierSegment cub)
03678                               {
03679                                   tbr.Segments.AddRange(cub.BreakAtInflectionPoints(currPoint,
      flatness));
03680                                   currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03681                               }
03682                               else
03683                               {
03684                                   tbr.Segments.Add(seg2);
03685                                   currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03686                               }
03687                           }
03688                           break;
03689                       case SegmentType.CubicBezier:
03690
      tbr.Segments.AddRange(((CubicBezierSegment)seg).BreakAtInflectionPoints(currPoint, flatness));
03691                           currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03692                           break;
03693                   }
03694               }
03695
03696               return tbr;
03697           }
03698
03699           private int ContainsPointInternal(Point point, bool evenOdd)
03700           {
03701               int countsAbs = 0;
03702               int counts = 0;
03703
03704               foreach (GraphicsPath figure in this.MonotoniseOnY().GetFigures())
03705               {
03706                   if (figure.Segments[figure.Segments.Count - 1].Type == SegmentType.Close)
03707                   {
03708                       if (figure.Segments[0].Type != SegmentType.Move)
03709                       {
03710                           throw new InvalidOperationException("Assertion failed:  the figure does not
      start with a move segment!");
03711                       }
03712
```

```
03713                       (int, int)[] pointDirections = new (int, int)[figure.Segments.Count - 1];
03714
03715                       Point startPoint = new Point();
03716                       Point currPoint = new Point();
03717                       int index = 0;
03718
03719                       for (int i = 0; i < figure.Segments.Count; i++)
03720                       {
03721                           if (figure.Segments[i].Type == SegmentType.Move)
03722                           {
03723                               currPoint = figure.Segments[i].Point;
03724                               startPoint = currPoint;
03725                           }
03726                           else if (figure.Segments[i].Type == SegmentType.Line)
03727                           {
03728                               int direction = Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03729
03730                               pointDirections[index] = (pointDirections[index].Item1, direction);
03731                               pointDirections[index + 1] = (direction, pointDirections[index +
      1].Item2);
03732
03733                               currPoint = figure.Segments[i].Point;
03734                               index++;
03735                           }
03736                           else if (figure.Segments[i].Type == SegmentType.Arc)
03737                           {
03738                               throw new InvalidOperationException("Path not properly monotonised!");
03739                           }
03740                           else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03741                           {
03742                               int direction = Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03743
03744                               pointDirections[index] = (pointDirections[index].Item1, direction);
03745                               pointDirections[index + 1] = (direction, pointDirections[index +
      1].Item2);
03746
03747                               currPoint = figure.Segments[i].Point;
03748                               index++;
03749                           }
03750                           else if (figure.Segments[i].Type == SegmentType.Close)
03751                           {
03752                               int direction = Math.Sign(startPoint.Y - currPoint.Y);
03753
03754                               pointDirections[index] = (pointDirections[index].Item1, direction);
03755                               pointDirections[0] = (direction, pointDirections[0].Item2);
03756                           }
03757                       }
03758
03759                       int[] directions = new int[pointDirections.Length * 2];
03760
03761                       for (int i = 0; i < pointDirections.Length; i++)
03762                       {
03763                           directions[2 * i] = pointDirections[i].Item1;
03764                           directions[2 * i + 1] = pointDirections[i].Item2;
03765                       }
03766
03767                       for (int i = 0; i < directions.Length; i++)
03768                       {
03769                           if (directions[i] == 0)
03770                           {
03771                               int nextDir = 0;
03772
03773                               int j = 0;
03774                               while (directions[(i + j) % directions.Length] == 0)
03775                               {
03776                                   j++;
03777                               }
03778                               nextDir = directions[(i + j) % directions.Length];
03779
03780                               int prevDir = 0;
03781
03782                               j = 0;
03783                               while (directions[(i - j) >= 0 ?  (i - j) :  (i - j + directions.Length)]
      == 0)
03784                               {
03785                                   j++;
03786                               }
03787                               prevDir = directions[(i - j) >= 0 ?  (i - j) :  (i - j +
      directions.Length)];
03788
03789                               if (prevDir == nextDir)
03790                               {
03791                                   directions[i] = prevDir;
03792                               }
03793                           }
03794                       }
03795
```

```
03796                     for (int i = 0; i < pointDirections.Length; i++)
03797                     {
03798                         pointDirections[i] = (directions[2 * i], directions[2 * i + 1]);
03799                     }
03800 
03801                     startPoint = new Point();
03802                     currPoint = new Point();
03803                     index = 0;
03804 
03805                     for (int i = 0; i < figure.Segments.Count; i++)
03806                     {
03807                         if (figure.Segments[i].Type == SegmentType.Move)
03808                         {
03809                             currPoint = figure.Segments[i].Point;
03810                             startPoint = currPoint;
03811                         }
03812                         else if (figure.Segments[i].Type == SegmentType.Line)
03813                         {
03814                             if (currPoint.Y != figure.Segments[i].Point.Y)
03815                             {
03816                                 double t = (point.Y - currPoint.Y) / (figure.Segments[i].Point.Y -
      currPoint.Y);
03817 
03818                                 if (t >= 0 && t <= 1)
03819                                 {
03820                                     double x = currPoint.X + (figure.Segments[i].Point.X -
      currPoint.X) * t;
03821 
03822                                     if (x >= point.X)
03823                                     {
03824                                         if (t == 0)
03825                                         {
03826                                             if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03827                                             {
03828                                                 countsAbs++;
03829                                                 counts += pointDirections[index].Item2;
03830                                             }
03831                                         }
03832                                         else if (t == 1)
03833                                         {
03834                                             if (!((pointDirections[index + 1].Item1 == 1 &&
      pointDirections[index + 1].Item2 == 1) || (pointDirections[index + 1].Item1 == 0 &&
      pointDirections[index + 1].Item2 == 0)))
03835                                             {
03836                                                 countsAbs++;
03837                                                 counts += pointDirections[index + 1].Item1;
03838                                             }
03839                                         }
03840                                         else
03841                                         {
03842                                             countsAbs++;
03843                                             counts += Math.Sign(figure.Segments[i].Point.Y -
      currPoint.Y);
03844                                         }
03845                                     }
03846                                 }
03847                             }
03848                             /* else if (currPoint.Y == point.Y)
03849 {
03850 if (currPoint.X >= point.X)
03851 {
03852 if (!((pointDirections[index].Item1 == -1 && pointDirections[index].Item2 == -1) ||
      (pointDirections[index].Item1 == 0 && pointDirections[index].Item2 == 0)))
03853 {
03854 countsAbs++;
03855 counts += pointDirections[index].Item2;
03856 }
03857 }
03858 
03859 if (figure.Segments[i].Point.X >= point.X)
03860 {
03861 if (!((pointDirections[index + 1].Item1 == 1 && pointDirections[index + 1].Item2 == 1) ||
      (pointDirections[index + 1].Item1 == 0 && pointDirections[index + 1].Item2 == 0)))
03862 {
03863 countsAbs++;
03864 counts += pointDirections[index + 1].Item1;
03865 }
03866 }
03867 }*/
03868 
03869                             currPoint = figure.Segments[i].Point;
03870                             index++;
03871                         }
03872                         else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03873                         {
```

```
03874                               double a = -currPoint.Y + 3 * figure.Segments[i].Points[0].Y - 3 *
      figure.Segments[i].Points[1].Y + figure.Segments[i].Points[2].Y;
03875                               double b = 3 * currPoint.Y - 6 * figure.Segments[i].Points[0].Y + 3 *
      figure.Segments[i].Points[1].Y;
03876                               double c = -3 * currPoint.Y + 3 * figure.Segments[i].Points[0].Y;
03877
03878
03879                               double d = currPoint.Y - point.Y;
03880
03881                               double[] roots = CubicRoots(a, b, c, d);
03882
03883                               if (roots.Length == 1)
03884                               {
03885                                   double t = roots[0];
03886                                   double x = (1 - t) * (1 - t) * (1 - t) * currPoint.X +
      figure.Segments[i].Points[0].X * t * (1 - t) * (1 - t) * 3 + figure.Segments[i].Points[1].X * t * t *
      (1 - t) * 3 + figure.Segments[i].Points[2].X * t * t * t;
03887
03888                                   if (x >= point.X)
03889                                   {
03890                                       if (t == 0)
03891                                       {
03892                                           if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03893                                           {
03894                                               countsAbs++;
03895                                               counts += pointDirections[index].Item2;
03896                                           }
03897                                       }
03898                                       else if (t == 1)
03899                                       {
03900                                           if (!((pointDirections[index + 1].Item1 == 1 &&
      pointDirections[index + 1].Item2 == 1) || (pointDirections[index + 1].Item1 == 0 &&
      pointDirections[index + 1].Item2 == 0)))
03901                                           {
03902                                               countsAbs++;
03903                                               counts += pointDirections[index + 1].Item1;
03904                                           }
03905                                       }
03906                                       else
03907                                       {
03908                                           countsAbs++;
03909                                           counts += Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03910                                       }
03911                                   }
03912                               }
03913                               else if (roots.Length != 0)
03914                               {
03915                                   throw new InvalidOperationException("Path not properly monotonised!");
03916                               }
03917
03918                               currPoint = figure.Segments[i].Point;
03919                               index++;
03920                           }
03921                           else if (figure.Segments[i].Type == SegmentType.Close)
03922                           {
03923                               if (currPoint.Y != startPoint.Y)
03924                               {
03925                                   double t = (point.Y - currPoint.Y) / (startPoint.Y - currPoint.Y);
03926
03927                                   if (t >= 0 && t <= 1)
03928                                   {
03929                                       double x = currPoint.X + (startPoint.X - currPoint.X) * t;
03930
03931                                       if (x >= point.X)
03932                                       {
03933                                           if (t == 0)
03934                                           {
03935                                               if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03936                                               {
03937                                                   countsAbs++;
03938                                                   counts += pointDirections[index].Item2;
03939                                               }
03940                                           }
03941                                           else if (t == 1)
03942                                           {
03943                                               if (!((pointDirections[0].Item1 == 1 &&
      pointDirections[0].Item2 == 1) || (pointDirections[0].Item1 == 0 && pointDirections[0].Item2 == 0)))
03944                                               {
03945                                                   countsAbs++;
03946                                                   counts += pointDirections[0].Item1;
03947                                               }
03948                                           }
03949                                           else
```

```
03950                                           {
03951                                               countsAbs++;
03952                                               counts += Math.Sign(startPoint.Y - currPoint.Y);
03953                                           }
03954                                       }
03955                                   }
03956                               }
03957                               /* else if (currPoint.Y == point.Y)
03958 {
03959 if (currPoint.X >= point.X)
03960 {
03961 if (!((pointDirections[index].Item1 == -1 && pointDirections[index].Item2 == -1) ||
      (pointDirections[index].Item1 == 0 && pointDirections[index].Item2 == 0)))
03962 {
03963 countsAbs++;
03964 counts += pointDirections[index].Item2;
03965 }
03966 }
03967
03968 if (startPoint.X >= point.X)
03969 {
03970 if (!((pointDirections[0].Item1 == 1 && pointDirections[0].Item2 == 1) || (pointDirections[0].Item1 ==
      0 && pointDirections[0].Item2 == 0)))
03971 {
03972 countsAbs++;
03973 counts += pointDirections[0].Item1;
03974 }
03975 }
03976 }*/
03977                           }
03978                       }
03979                   }
03980               }
03981
03982               if (evenOdd)
03983               {
03984                   return countsAbs % 2;
03985               }
03986               else
03987               {
03988                   return counts != 0 ?  1 :  0;
03989               }
03990           }
03991
03992           private static void ProcessHalfStroke(GraphicsPath figure, GraphicsPath tbr, double lineWidth,
      LineJoins lineJoin, double flatness, bool positive)
03993           {
03994               Point currPoint = new Point();
03995               Point currTangent = new Point(double.NaN, double.NaN);
03996               double sign = positive ?  1 :  -1;
03997
03998               bool started = false;
03999               Point startPoint = new Point();
04000
04001               bool isClosed = false;
04002
04003               figure = RemoveDuplicates(figure.Monotonise(flatness));
04004
04005               foreach (Segment seg in figure.Segments)
04006               {
04007                   if (seg.Type == SegmentType.Move)
04008                   {
04009                       currPoint = seg.Point;
04010                       currTangent = new Point(double.NaN, double.NaN);
04011                       startPoint = currPoint;
04012                       started = true;
04013                   }
04014                   else if (seg.Type == SegmentType.Line)
04015                   {
04016                       Point segStartPoint = currPoint;
04017                       Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04018                       Point segEndPoint = seg.Point;
04019
04020                       if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04021                       {
04022                           Point p = currPoint;
04023                           Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04024                           Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04025
04026                           Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04027                           Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
```

```
04028
04029                           Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04030                           Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04031
04032                           double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04033
04034                           if (crossProduct > 0 && !positive)
04035                           {
04036                               if (lineJoin == LineJoins.Round)
04037                               {
04038                                   tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04039                               }
04040                               else if (lineJoin == LineJoins.Miter)
04041                               {
04042                                   tbr.LineTo(p3_1);
04043                               }
04044                           }
04045                           else if (crossProduct < 0 && positive)
04046                           {
04047                               if (lineJoin == LineJoins.Round)
04048                               {
04049                                   tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04050                               }
04051                               else if (lineJoin == LineJoins.Miter)
04052                               {
04053                                   tbr.LineTo(p3_2);
04054                               }
04055                           }
04056                       }
04057
04058                       if (!double.IsNaN(segTangent.X) && !double.IsNaN(segTangent.Y))
04059                       {
04060                           tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04061                           tbr.LineTo(seg.Point + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04062                       }
04063
04064                       currPoint = seg.Point;
04065                       currTangent = segTangent;
04066
04067                       if (!started)
04068                       {
04069                           startPoint = currPoint;
04070                           started = true;
04071                       }
04072                   }
04073                   else if (seg.Type == SegmentType.Arc && seg is ArcSegment arc)
04074                   {
04075                       Point segStartPoint = currPoint;
04076                       Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04077                       Point segEndPoint = seg.Point;
04078
04079                       if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04080                       {
04081                           Point p = currPoint;
04082                           Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04083                           Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04084
04085                           Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04086                           Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04087
04088                           Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04089                           Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04090
04091                           double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04092                           if (crossProduct > 0 && !positive)
04093                           {
04094                               if (lineJoin == LineJoins.Round)
04095                               {
04096                                   tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04097                               }
04098                               else if (lineJoin == LineJoins.Miter)
04099                               {
04100                                   tbr.LineTo(p3_1);
04101                               }
```

```
04102                         }
04103                         else if (crossProduct < 0 && positive)
04104                         {
04105                             if (lineJoin == LineJoins.Round)
04106                             {
04107                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04108                             }
04109                             else if (lineJoin == LineJoins.Miter)
04110                             {
04111                                 tbr.LineTo(p3_2);
04112                             }
04113                         }
04114                     }
04115 
04116                     tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04117                     tbr.Arc(arc.Points[0], arc.Radius - sign * lineWidth * 0.5 *
      Math.Sign(arc.EndAngle - arc.StartAngle), arc.StartAngle, arc.EndAngle);
04118 
04119                     currPoint = seg.Point;
04120                     currTangent = seg.GetTangentAt(segStartPoint, 1);
04121 
04122                     if (!started)
04123                     {
04124                         startPoint = currPoint;
04125                         started = true;
04126                     }
04127                 }
04128                 else if (seg.Type == SegmentType.CubicBezier)
04129                 {
04130                     Point segStartPoint = currPoint;
04131                     Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04132                     Point segEndPoint = seg.Point;
04133 
04134                     if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04135                     {
04136                         Point p = currPoint;
04137                         Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04138                         Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04139 
04140                         Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04141                         Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04142 
04143                         Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04144                         Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04145 
04146                         double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04147 
04148                         if (crossProduct > 1e-7 && !positive)
04149                         {
04150                             if (lineJoin == LineJoins.Round)
04151                             {
04152                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04153                             }
04154                             else if (lineJoin == LineJoins.Miter)
04155                             {
04156                                 tbr.LineTo(p3_1);
04157                             }
04158                         }
04159                         else if (crossProduct < -1e-7 && positive)
04160                         {
04161                             if (lineJoin == LineJoins.Round)
04162                             {
04163                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04164                             }
04165                             else if (lineJoin == LineJoins.Miter)
04166                             {
04167                                 tbr.LineTo(p3_2);
04168                             }
04169                         }
04170                     }
04171 
04172                     if (!double.IsNaN(segTangent.X) && !double.IsNaN(segTangent.Y))
04173                     {
04174                         tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04175                     }
```

```
04176
04177                     List<(Point point, Point tangent)> newPoints =
    seg.FlattenForOffsetAndGetTangents(currPoint, sign * lineWidth, flatness).ToList();
04178
04179                     for (int i = 0; i < newPoints.Count; i++)
04180                     {
04181                         Point pt = newPoints[i].point + sign * lineWidth * 0.5 * new
    Point(-newPoints[i].tangent.Y, newPoints[i].tangent.X);
04182
04183                         if (!double.IsNaN(pt.X) && !double.IsNaN(pt.Y))
04184                         {
04185                             tbr.LineTo(pt);
04186                         }
04187                     }
04188
04189                     if ((currPoint.X != newPoints[newPoints.Count - 1].point.X || currPoint.Y !=
    newPoints[newPoints.Count - 1].point.Y) || double.IsNaN(currTangent.X) || double.IsNaN(currTangent.Y))
04190                     {
04191                         currPoint = newPoints[newPoints.Count - 1].point;
04192                         currTangent = newPoints[newPoints.Count - 1].tangent;
04193                     }
04194
04195                     if (!started)
04196                     {
04197                         startPoint = currPoint;
04198                         started = true;
04199                     }
04200                 }
04201                 else if (seg.Type == SegmentType.Close)
04202                 {
04203                     Point segStartPoint = currPoint;
04204                     Point segTangent = new Point(startPoint.X - currPoint.X, startPoint.Y -
    currPoint.Y).Normalize();
04205                     Point segEndPoint = startPoint;
04206
04207                     if (!currPoint.IsEqual(startPoint, Tolerance))
04208                     {
04209                         if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) &&
    (currTangent.X != segTangent.X && currTangent.Y != segTangent.Y))
04210                         {
04211                             Point p = currPoint;
04212                             Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
    lineWidth * 0.5;
04213                             Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
    lineWidth * 0.5;
04214
04215                             Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) *
    lineWidth * 0.5;
04216                             Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) *
    lineWidth * 0.5;
04217
04218                             Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04219                             Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04220
04221                             double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
    segTangent.X;
04222
04223                             if (crossProduct > 0 && !positive)
04224                             {
04225                                 if (lineJoin == LineJoins.Round)
04226                                 {
04227                                     tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
    p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04228                                 }
04229                                 else if (lineJoin == LineJoins.Miter)
04230                                 {
04231                                     tbr.LineTo(p3_1);
04232                                 }
04233                             }
04234                             else if (crossProduct < 0 && positive)
04235                             {
04236                                 if (lineJoin == LineJoins.Round)
04237                                 {
04238                                     tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X
    - p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04239                                 }
04240                                 else if (lineJoin == LineJoins.Miter)
04241                                 {
04242                                     tbr.LineTo(p3_2);
04243                                 }
04244                             }
04245                         }
04246
04247
04248                         tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
    segTangent.X));
04249
```

```
04250                         tbr.LineTo(startPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04251
04252                         currTangent = segTangent;
04253                     }
04254
04255                     currPoint = startPoint;
04256
04257
04258                     isClosed = true;
04259
04260                     started = false;
04261                 }
04262             }
04263
04264
04265             if (isClosed)
04266             {
04267                 Point segTangent = figure.GetTangentAtRelative(0);
04268
04269                 if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X !=
      segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04270                 {
04271                     Point p = currPoint;
04272                     Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) * lineWidth *
      0.5;
04273                     Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) * lineWidth *
      0.5;
04274
04275                     Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth *
      0.5;
04276                     Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth *
      0.5;
04277
04278                     Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04279                     Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04280
04281                     double crossProduct = currTangent.X * segTangent.Y - currTangent.Y * segTangent.X;
04282
04283                     if (crossProduct > 0 && !positive)
04284                     {
04285                         if (lineJoin == LineJoins.Round)
04286                         {
04287                             double ang1 = Math.Atan2(pALeft.Y - p.Y, pALeft.X - p.X);
04288                             double ang2 = Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X);
04289
04290                             if (ang1 < 0)
04291                             {
04292                                 ang1 += 2 * Math.PI;
04293                             }
04294
04295                             if (ang2 < 0)
04296                             {
04297                                 ang2 += 2 * Math.PI;
04298                             }
04299
04300                             tbr.Arc(p, lineWidth * 0.5, ang1, ang2);
04301                         }
04302                         else if (lineJoin == LineJoins.Miter)
04303                         {
04304                             tbr.LineTo(p3_1);
04305                         }
04306                     }
04307                     else if (crossProduct < 0 && positive)
04308                     {
04309                         if (lineJoin == LineJoins.Round)
04310                         {
04311                             double ang1 = Math.Atan2(pARight.Y - p.Y, pARight.X - p.X);
04312                             double ang2 = Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X);
04313
04314                             if (ang1 < 0)
04315                             {
04316                                 ang1 += 2 * Math.PI;
04317                             }
04318
04319                             if (ang2 < 0)
04320                             {
04321                                 ang2 += 2 * Math.PI;
04322                             }
04323
04324                             tbr.Arc(p, lineWidth * 0.5, ang1, ang2);
04325                         }
04326                         else if (lineJoin == LineJoins.Miter)
04327                         {
04328                             tbr.LineTo(p3_2);
04329                         }
```

```
04330                     }
04331                 }
04332
04333
04334                 tbr.Close();
04335             }
04336         }
04337
04338         internal static double Tolerance = 1e-7;
04339
04340         private static GraphicsPath RemoveDuplicates(GraphicsPath path)
04341         {
04342             Point currentPoint = new Point();
04343             Point startPoint = new Point();
04344             bool started = false;
04345
04346             GraphicsPath tbr = new GraphicsPath();
04347
04348             GraphicsPath figure = null;
04349
04350             for (int i = 0; i < path.Segments.Count; i++)
04351             {
04352                 if (path.Segments[i].Type == SegmentType.Move)
04353                 {
04354                     if (figure != null)
04355                     {
04356                         tbr.AddPath(figure);
04357                     }
04358
04359                     figure = new GraphicsPath();
04360
04361                     currentPoint = path.Segments[i].Point;
04362                     figure.MoveTo(currentPoint);
04363                     startPoint = currentPoint;
04364                     started = true;
04365                 }
04366                 else if (path.Segments[i].Type == SegmentType.Line)
04367                 {
04368                     if (figure == null)
04369                     {
04370                         figure = new GraphicsPath();
04371                     }
04372
04373                     if (currentPoint.X != path.Segments[i].Point.X || currentPoint.Y !=
      path.Segments[i].Point.Y)
04374                     {
04375                         currentPoint = path.Segments[i].Point;
04376                         figure.LineTo(currentPoint);
04377                     }
04378
04379                     if (!started)
04380                     {
04381                         startPoint = currentPoint;
04382                         started = true;
04383                     }
04384                 }
04385                 else if (path.Segments[i].Type == SegmentType.Arc && path.Segments[i] is ArcSegment
      arc)
04386                 {
04387                     if (figure == null)
04388                     {
04389                         figure = new GraphicsPath();
04390                     }
04391
04392                     Point endPoint = arc.GetPointAt(currentPoint, 1);
04393
04394                     if (arc.StartAngle != arc.EndAngle)
04395                     {
04396                         currentPoint = endPoint;
04397                         figure.Arc(arc.Points[0], arc.Radius, arc.StartAngle, arc.EndAngle);
04398                     }
04399                     else if (endPoint.X != currentPoint.X || endPoint.Y != currentPoint.Y)
04400                     {
04401                         currentPoint = endPoint;
04402                         figure.LineTo(endPoint);
04403                     }
04404
04405                     if (!started)
04406                     {
04407                         startPoint = currentPoint;
04408                         started = true;
04409                     }
04410                 }
04411                 else if (path.Segments[i].Type == SegmentType.CubicBezier)
04412                 {
04413                     if (figure == null)
04414                     {
```

```
04415                           figure = new GraphicsPath();
04416                       }
04417
04418                       if (!currentPoint.IsEqual(path.Segments[i].Points[0], Tolerance) ||
      !currentPoint.IsEqual(path.Segments[i].Points[1], Tolerance) ||
      !currentPoint.IsEqual(path.Segments[i].Points[2], Tolerance))
04419                       {
04420                           currentPoint = path.Segments[i].Points[2];
04421                           figure.CubicBezierTo(path.Segments[i].Points[0], path.Segments[i].Points[1],
      path.Segments[i].Points[2]);
04422                       }
04423
04424                       if (!started)
04425                       {
04426                           startPoint = currentPoint;
04427                           started = true;
04428                       }
04429                   }
04430                   else if (path.Segments[i].Type == SegmentType.Close)
04431                   {
04432                       if (currentPoint.X != startPoint.X || currentPoint.Y != startPoint.Y)
04433                       {
04434                           currentPoint = startPoint;
04435                           figure.Close();
04436
04437                           tbr.AddPath(figure);
04438                           figure = null;
04439                       }
04440                       else
04441                       {
04442                           if (figure.Segments[figure.Segments.Count - 1].Type == SegmentType.Line)
04443                           {
04444                               figure.Segments.RemoveAt(figure.Segments.Count - 1);
04445                           }
04446                           else if (figure.Segments[0].Type == SegmentType.Move &&
      figure.Segments[1].Type == SegmentType.Line)
04447                           {
04448                               figure.Segments[0] = new MoveSegment(figure.Segments[1].Point);
04449                               figure.Segments.RemoveAt(1);
04450                           }
04451
04452                           currentPoint = startPoint;
04453                           figure.Close();
04454
04455                           tbr.AddPath(figure);
04456                           figure = null;
04457                       }
04458
04459                       started = false;
04460                   }
04461               }
04462
04463               if (figure != null)
04464               {
04465                   tbr.AddPath(figure);
04466               }
04467
04468               return tbr;
04469           }
04470       }
04471   }
```

8.70 ImageFormats.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
```

```
00020 using System.IO;
00021 using System.IO.Compression;
00022 using System.Runtime.InteropServices;
00023 using System.Threading;
00024
00025 namespace VectSharp
00026 {
00027 /// <summary>
00028 /// Contains methods to create animated PNG image files.
00029 /// </summary>
00030     public static class AnimatedPNG
00031     {
00032 /// <summary>
00033 /// Types of inter-frame compression.
00034 /// </summary>
00035         public enum InterframeCompression
00036         {
00037 /// <summary>
00038 /// No inter-frame compression is performed (fastest, but produces the largest files).
00039 /// </summary>
00040             None,
00041
00042 /// <summary>
00043 /// Inter-frame compression is performed by encoding the difference between each frame and the first
      frame in the animation.
00044 /// </summary>
00045             First,
00046
00047 /// <summary>
00048 /// Inter-frame compression is performed by encoding the difference between each frame and the
      previous frame (produces the smallest files, but it is the slowest and requires large amounts of
      memory).
00049 /// </summary>
00050             Previous
00051         }
00052
00053 /// <summary>
00054 /// Represents an individual frame of a PNG animation.
00055 /// </summary>
00056         public class CompressedFrame :  IDisposable
00057         {
00058             internal MemoryStream compressedStream;
00059             private bool disposedValue;
00060
00061 /// <summary>
00062 /// The duration of the frame in milliseconds.
00063 /// </summary>
00064             public double Duration { get; }
00065
00066             internal InterframeCompression InterframeCompression { get; } =
      InterframeCompression.None;
00067
00068 /// <summary>
00069 /// Creates a new <see cref="CompressedFrame"/> from raw pixel data in RGB or RGBA format.
00070 /// </summary>
00071 /// <param name="rawFrameData">A pointer to the raw pixel data for the frame.</param>
00072 /// <param name="width">The width of the image in pixels.</param>
00073 /// <param name="height">The height of the image in pixels.</param>
00074 /// <param name="hasAlpha">If the image data is in RGBA format, set this to <see langword="true"/>; if
      it is in RGB format, set it to <see langword="false"/>.</param>
00075 /// <param name="duration">The duration of the frame in milliseconds.</param>
00076             public unsafe CompressedFrame(DisposableIntPtr rawFrameData, int width, int height, bool
      hasAlpha, double duration)
00077             {
00078                 this.compressedStream =
      PNGEncoder.CompressAPNGFrame((byte*)rawFrameData.InternalPointer, width, height, hasAlpha,
      PNGEncoder.FilterModes.Adaptive, 1);
00079                 this.Duration = duration;
00080                 this.InterframeCompression = InterframeCompression.None;
00081             }
00082
00083 /// <summary>
00084 /// Creates a new <see cref="CompressedFrame"/> from raw pixel data in RGB or RGBA format, applying
      inter-frame compression based on another frame.
00085 /// </summary>
00086 /// <param name="rawFrameData">A pointer to the raw pixel data for the new frame.</param>
00087 /// <param name="otherFrameData">A pointer to the raw pixel data for the frame to use as a reference.
      This can either be the first frame in the animation, or the frame immediately preceeding the current
      frame.</param>
00088 /// <param name="isFirstFrame">If <paramref name="otherFrameData"/> is the raw pixel data for the
      first frame in the animation, set this to <see langword="true"/>; if it is the immediately preceeding
      frame, set this to <see langword="false"/>.</param>
00089 /// <param name="width">The width of the image in pixels.</param>
00090 /// <param name="height">The height of the image in pixels.</param>
00091 /// <param name="hasAlpha">If the image data is in RGBA format, set this to <see langword="true"/>; if
      it is in RGB format, set it to <see langword="false"/>.</param>
00092 /// <param name="duration">The duration of the frame in milliseconds.</param>
```

```
00093             public unsafe CompressedFrame(DisposableIntPtr rawFrameData, DisposableIntPtr
      otherFrameData, bool isFirstFrame, int width, int height, bool hasAlpha, double duration)
00094             {
00095                 int pixelSize = hasAlpha ?  4 :  3;
00096 
00097                 IntPtr frameDiff = Marshal.AllocHGlobal(width * height * 4);
00098 
00099                 byte* frameDiffData = (byte*)frameDiff;
00100                 byte* previousFrame = (byte*)otherFrameData.InternalPointer;
00101                 byte* currentFrame = (byte*)rawFrameData.InternalPointer;
00102 
00103                 for (int y = 0; y < height; y++)
00104                 {
00105                     for (int x = 0; x < width; x++)
00106                     {
00107                         int pixelIndex = (y * width + x) * pixelSize;
00108 
00109                         if (previousFrame[pixelIndex] != currentFrame[pixelIndex] ||
      previousFrame[pixelIndex + 1] != currentFrame[pixelIndex + 1] ||
00110                             previousFrame[pixelIndex + 2] != currentFrame[pixelIndex + 2] || (hasAlpha
      && previousFrame[pixelIndex + 3] != currentFrame[pixelIndex + 3]))
00111                         {
00112                             frameDiffData[pixelIndex] = currentFrame[pixelIndex];
00113                             frameDiffData[pixelIndex + 1] = currentFrame[pixelIndex + 1];
00114                             frameDiffData[pixelIndex + 2] = currentFrame[pixelIndex + 2];
00115 
00116                             if (hasAlpha)
00117                             {
00118                                 frameDiffData[pixelIndex + 3] = currentFrame[pixelIndex + 3];
00119                             }
00120                             else
00121                             {
00122                                 frameDiffData[pixelIndex + 3] = 255;
00123                             }
00124                         }
00125                         else
00126                         {
00127                             frameDiffData[pixelIndex] = 0;
00128                             frameDiffData[pixelIndex + 1] = 0;
00129                             frameDiffData[pixelIndex + 2] = 0;
00130                             frameDiffData[pixelIndex + 3] = 0;
00131                         }
00132                     }
00133                 }
00134 
00135                 this.compressedStream = PNGEncoder.CompressAPNGFrame((byte*)frameDiff, width, height,
      true, PNGEncoder.FilterModes.Adaptive, 1);
00136                 this.Duration = duration;
00137 
00138                 if (isFirstFrame)
00139                 {
00140                     this.InterframeCompression = InterframeCompression.First;
00141                 }
00142                 else
00143                 {
00144                     this.InterframeCompression = InterframeCompression.Previous;
00145                 }
00146 
00147                 Marshal.FreeHGlobal(frameDiff);
00148             }
00149 
00150 /// <inheritdoc/>
00151             protected virtual void Dispose(bool disposing)
00152             {
00153                 if (!disposedValue)
00154                 {
00155                     if (disposing)
00156                     {
00157                         this.compressedStream.Dispose();
00158                     }
00159 
00160                     disposedValue = true;
00161                 }
00162             }
00163 
00164 /// <inheritdoc/>
00165             public void Dispose()
00166             {
00167                 Dispose(disposing:  true);
00168                 GC.SuppressFinalize(this);
00169             }
00170         }
00171 
00172 /// <summary>
00173 /// Create a new animated PNG image, outputting it to the specified stream.
00174 /// </summary>
00175 /// <param name="outputStream">The stream to which the animated PNG image will be written.</param>
```

```
00176 /// <param name="width">The width of the image in pixels.</param>
00177 /// <param name="height">The height of the image in pixels.</param>
00178 /// <param name="hasAlpha">If the frames of the image have an alpha channel, set this to <see
      langword="true"/>; otherwise, set it to <see langword="false"/>.</param>
00179 /// <param name="compressedFrames">The frames that will be used to create the animated PNG
      image.</param>
00180 /// <param name="repeatCount">The number of times that the animation should loop.  Set this to 0 for
      an infinitely repeating animation.</param>
00181         public static unsafe void Create(Stream outputStream, int width, int height, bool hasAlpha,
      IReadOnlyList<CompressedFrame> compressedFrames, int repeatCount)
00182         {
00183             PNGEncoder.StartAPNG(width, height, hasAlpha, outputStream, compressedFrames.Count,
      repeatCount);
00184 
00185             for (int i = 0; i < compressedFrames.Count; i++)
00186             {
00187                 ushort frameDurationNumerator = (ushort)Math.Round(compressedFrames[i].Duration * 60);
00188                 ushort frameDurationDenominator = 60000;
00189                 PNGEncoder.AddPreCompressedAPNGFrame(compressedFrames[i], width, height, hasAlpha,
      outputStream, i, frameDurationNumerator, frameDurationDenominator);
00190             }
00191 
00192             PNGEncoder.FinishAPNG(outputStream);
00193         }
00194     }
00195 
00196 
00197 
00198     internal static class PNGEncoder
00199     {
00200         internal unsafe static void SavePNG(byte* image, int width, int height, bool hasAlpha, Stream
      fs, FilterModes filter, int threadCount = 0)
00201         {
00202             if (threadCount == 0)
00203             {
00204                 threadCount = filter == FilterModes.Adaptive ?  Math.Max(1, Math.Min(width / 600,
      Environment.ProcessorCount - 2)) :  1;
00205             }
00206 
00207             //Header
00208             fs.Write(new byte[] { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A }, 0, 8);
00209 
00210             //IHDR chunk
00211             fs.WriteInt(13);
00212             using (MemoryStream ihdr = new MemoryStream(13))
00213             {
00214                 ihdr.WriteASCIIString("IHDR");
00215                 ihdr.WriteInt(width);
00216                 ihdr.WriteInt(height);
00217                 ihdr.WriteByte(8); //Bit depth
00218 
00219                 if (hasAlpha)
00220                 {
00221                     ihdr.WriteByte(6); //Colour type
00222                 }
00223                 else
00224                 {
00225                     ihdr.WriteByte(2); //Colour type
00226                 }
00227 
00228                 ihdr.WriteByte(0); //Compression method
00229                 ihdr.WriteByte(0); //Filter method
00230                 ihdr.WriteByte(0); //Interlace
00231 
00232                 ihdr.Seek(0, SeekOrigin.Begin);
00233                 ihdr.CopyTo(fs);
00234 
00235                 fs.WriteUInt(CRC32.ComputeCRC(ihdr));
00236             }
00237 
00238             //IDAT chunk
00239             IntPtr filteredImage;
00240 
00241             if (threadCount > 1)
00242             {
00243                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00244             }
00245             else
00246             {
00247                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00248             }
00249 
00250             using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00251             {
00252                 compressedImage.Seek(0, SeekOrigin.Begin);
```

```
00253                 fs.WriteUInt((uint)compressedImage.Length);
00254                 fs.WriteASCIIString("IDAT");
00255                 compressedImage.Seek(0, SeekOrigin.Begin);
00256                 compressedImage.CopyTo(fs);
00257 
00258                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00259             }
00260 
00261             Marshal.FreeHGlobal(filteredImage);
00262 
00263             //IEND chunk
00264             fs.WriteInt(0);
00265             fs.WriteASCIIString("IEND");
00266             fs.Write(new byte[] { 0xAE, 0x42, 0x60, 0x82 }, 0, 4);
00267 
00268         }
00269 
00270         public unsafe static void StartAPNG(int width, int height, bool hasAlpha, Stream fs, int
      frameCount, int repeatCount)
00271         {
00272             //Header
00273             fs.Write(new byte[] { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A }, 0, 8);
00274 
00275             //IHDR chunk
00276             fs.WriteInt(13);
00277             using (MemoryStream ihdr = new MemoryStream(17))
00278             {
00279                 ihdr.WriteASCIIString("IHDR");
00280                 ihdr.WriteInt(width);
00281                 ihdr.WriteInt(height);
00282                 ihdr.WriteByte(8); //Bit depth
00283 
00284                 if (hasAlpha)
00285                 {
00286                     ihdr.WriteByte(6); //Colour type
00287                 }
00288                 else
00289                 {
00290                     ihdr.WriteByte(2); //Colour type
00291                 }
00292 
00293                 ihdr.WriteByte(0); //Compression method
00294                 ihdr.WriteByte(0); //Filter method
00295                 ihdr.WriteByte(0); //Interlace
00296 
00297                 ihdr.Seek(0, SeekOrigin.Begin);
00298                 ihdr.CopyTo(fs);
00299 
00300                 fs.WriteUInt(CRC32.ComputeCRC(ihdr));
00301             }
00302 
00303             //acTL chunk
00304             fs.WriteInt(8);
00305             using (MemoryStream actl = new MemoryStream(12))
00306             {
00307                 actl.WriteASCIIString("acTL");
00308                 actl.WriteInt(frameCount);
00309                 actl.WriteInt(repeatCount);
00310 
00311                 actl.Seek(0, SeekOrigin.Begin);
00312                 actl.CopyTo(fs);
00313 
00314                 fs.WriteUInt(CRC32.ComputeCRC(actl));
00315             }
00316         }
00317 
00318         internal unsafe static void AddAPNGFrame(byte* image, int width, int height, bool hasAlpha,
      Stream fs, FilterModes filter, int frameNumber, ushort frameDelayNumerator, ushort
      frameDelayDenominator, int threadCount = 0)
00319         {
00320             if (threadCount == 0)
00321             {
00322                 threadCount = filter == FilterModes.Adaptive ?  Math.Max(1, Math.Min(width / 600,
      Environment.ProcessorCount - 2)) :  1;
00323             }
00324 
00325             int fcTLIndex = frameNumber == 0 ?  0 :  ((frameNumber - 1) * 2 + 1);
00326             int fDATIndex = frameNumber == 0 ?  0 :  (frameNumber * 2);
00327 
00328             //fcTL chunk
00329             fs.WriteInt(26);
00330             using (MemoryStream fctl = new MemoryStream(30))
00331             {
00332                 fctl.WriteASCIIString("fcTL");
00333                 fctl.WriteInt(fcTLIndex);
00334                 fctl.WriteInt(width);
```

```
00335                 fctl.WriteInt(height);
00336                 fctl.WriteInt(0);
00337                 fctl.WriteInt(0);
00338                 fctl.WriteUShort(frameDelayNumerator);
00339                 fctl.WriteUShort(frameDelayDenominator);
00340                 fctl.WriteByte(1);
00341                 fctl.WriteByte(0);
00342 
00343                 fctl.Seek(0, SeekOrigin.Begin);
00344                 fctl.CopyTo(fs);
00345 
00346                 fs.WriteUInt(CRC32.ComputeCRC(fctl));
00347             }
00348 
00349             //fDAT chunk
00350             IntPtr filteredImage;
00351 
00352             if (threadCount > 1)
00353             {
00354                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00355             }
00356             else
00357             {
00358                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00359             }
00360 
00361             if (frameNumber > 0)
00362             {
00363                 using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00364                 {
00365                     fs.WriteUInt((uint)compressedImage.Length + 4);
00366 
00367                     fs.WriteASCIIString("fdAT");
00368                     fs.WriteInt(fDATIndex);
00369 
00370                     compressedImage.Seek(0, SeekOrigin.Begin);
00371                     compressedImage.CopyTo(fs);
00372 
00373                     fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 102, 100, 65, 84, (byte)(fDATIndex » 24), (byte)((fDATIndex
      » 16) & 255), (byte)((fDATIndex » 8) & 255), (byte)(fDATIndex & 255) }));
00374                     //fs.WriteUInt(CRC32.ComputeCRC(chunkStream.GetBuffer(), (int)chunkStream.Length,
      new byte[] { 73, 68, 65, 84 }));
00375                     //fs.WriteUInt(CRC32.ComputeCRC(chunkStream));
00376                 }
00377             }
00378             else
00379             {
00380                 using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00381                 {
00382                     compressedImage.Seek(0, SeekOrigin.Begin);
00383                     fs.WriteUInt((uint)compressedImage.Length);
00384                     fs.WriteASCIIString("IDAT");
00385                     compressedImage.Seek(0, SeekOrigin.Begin);
00386                     compressedImage.CopyTo(fs);
00387 
00388                     fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00389                 }
00390             }
00391 
00392             Marshal.FreeHGlobal(filteredImage);
00393         }
00394 
00395         internal static unsafe MemoryStream CompressAPNGFrame(byte* image, int width, int height, bool
      hasAlpha, FilterModes filter, int threadCount)
00396         {
00397             //fDAT chunk
00398             IntPtr filteredImage;
00399 
00400             if (threadCount > 1)
00401             {
00402                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00403             }
00404             else
00405             {
00406                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00407             }
00408 
00409             MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height * (width *
      (hasAlpha ?  4 :  3) + 1));
00410 
00411             Marshal.FreeHGlobal(filteredImage);
```

```
00412
00413             return compressedImage;
00414         }
00415
00416         internal static void AddPreCompressedAPNGFrame(AnimatedPNG.CompressedFrame compressedFrame,
      int width, int height, bool hasAlpha, Stream fs, int frameNumber, ushort frameDelayNumerator, ushort
      frameDelayDenominator)
00417         {
00418             int fcTLIndex = frameNumber == 0 ?  0 :  ((frameNumber - 1) * 2 + 1);
00419             int fDATIndex = frameNumber == 0 ?  0 :  (frameNumber * 2);
00420
00421             //fcTL chunk
00422             fs.WriteInt(26);
00423             using (MemoryStream fctl = new MemoryStream(30))
00424             {
00425                 fctl.WriteASCIIString("fcTL");
00426                 fctl.WriteInt(fcTLIndex);
00427                 fctl.WriteInt(width);
00428                 fctl.WriteInt(height);
00429                 fctl.WriteInt(0);
00430                 fctl.WriteInt(0);
00431                 fctl.WriteUShort(frameDelayNumerator);
00432                 fctl.WriteUShort(frameDelayDenominator);
00433
00434                 if (compressedFrame.InterframeCompression == AnimatedPNG.InterframeCompression.None ||
      frameNumber == 0)
00435                 {
00436                     fctl.WriteByte(0);
00437                     fctl.WriteByte(0);
00438                 }
00439                 else if (compressedFrame.InterframeCompression ==
      AnimatedPNG.InterframeCompression.First)
00440                 {
00441                     fctl.WriteByte(2);
00442                     fctl.WriteByte(1);
00443                 }
00444                 else if (compressedFrame.InterframeCompression ==
      AnimatedPNG.InterframeCompression.Previous)
00445                 {
00446                     fctl.WriteByte(0);
00447                     fctl.WriteByte(1);
00448                 }
00449
00450                 fctl.Seek(0, SeekOrigin.Begin);
00451                 fctl.CopyTo(fs);
00452
00453                 fs.WriteUInt(CRC32.ComputeCRC(fctl));
00454             }
00455
00456             MemoryStream compressedImage = compressedFrame.compressedStream;
00457
00458             if (frameNumber > 0)
00459             {
00460                 fs.WriteUInt((uint)compressedImage.Length + 4);
00461
00462                 fs.WriteASCIIString("fdAT");
00463                 fs.WriteInt(fDATIndex);
00464
00465                 compressedImage.Seek(0, SeekOrigin.Begin);
00466                 compressedImage.CopyTo(fs);
00467
00468                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 102, 100, 65, 84, (byte)(fDATIndex » 24), (byte)((fDATIndex
      » 16) & 255), (byte)((fDATIndex » 8) & 255), (byte)(fDATIndex & 255) }));
00469             }
00470             else
00471             {
00472                 compressedImage.Seek(0, SeekOrigin.Begin);
00473                 fs.WriteUInt((uint)compressedImage.Length);
00474                 fs.WriteASCIIString("IDAT");
00475                 compressedImage.Seek(0, SeekOrigin.Begin);
00476                 compressedImage.CopyTo(fs);
00477
00478                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00479             }
00480         }
00481
00482         internal unsafe static void FinishAPNG(Stream fs)
00483         {
00484             //IEND chunk
00485             fs.WriteInt(0);
00486             fs.WriteASCIIString("IEND");
00487             fs.Write(new byte[] { 0xAE, 0x42, 0x60, 0x82 }, 0, 4);
00488         }
00489
00490         internal enum FilterModes
```

```
00491         {
00492             None = 0,
00493             Sub = 1,
00494             Up = 2,
00495             Average = 3,
00496             Paeth = 4,
00497             Adaptive = -1
00498         }
00499 
00500         internal unsafe static void FilterRow(byte* image, int width, int pixelSize, int stride, int
      y, FilterModes filter, byte* destinationImage, byte* tempBuffer)
00501         {
00502             int startIndex = y * stride;
00503             int destinationIndex = y * (stride + 1);
00504 
00505             if (filter == FilterModes.None)
00506             {
00507                 destinationImage[destinationIndex++] = 0;
00508                 for (int x = 0; x < width; x++)
00509                 {
00510                     destinationImage[destinationIndex++] = image[startIndex++];
00511                     destinationImage[destinationIndex++] = image[startIndex++];
00512                     destinationImage[destinationIndex++] = image[startIndex++];
00513 
00514                     if (pixelSize == 4)
00515                     {
00516                         destinationImage[destinationIndex++] = image[startIndex++];
00517                     }
00518 
00519                 }
00520             }
00521             else if (filter == FilterModes.Sub)
00522             {
00523                 destinationImage[destinationIndex++] = 1;
00524                 for (int x = 0; x < width; x++)
00525                 {
00526                     if (x > 0)
00527                     {
00528                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00529                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00530                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00531                         if (pixelSize == 4)
00532                         {
00533                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00534                         }
00535                     }
00536                     else
00537                     {
00538                         destinationImage[destinationIndex++] = image[startIndex++];
00539                         destinationImage[destinationIndex++] = image[startIndex++];
00540                         destinationImage[destinationIndex++] = image[startIndex++];
00541                         if (pixelSize == 4)
00542                         {
00543                             destinationImage[destinationIndex++] = image[startIndex++];
00544                         }
00545                     }
00546                 }
00547             }
00548             else if (filter == FilterModes.Up)
00549             {
00550                 destinationImage[destinationIndex++] = 2;
00551                 for (int x = 0; x < width; x++)
00552                 {
00553                     if (y > 0)
00554                     {
00555                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00556                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00557                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00558                         if (pixelSize == 4)
00559                         {
00560                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00561                         }
00562                     }
00563                     else
00564                     {
00565                         destinationImage[destinationIndex++] = image[startIndex++];
00566                         destinationImage[destinationIndex++] = image[startIndex++];
00567                         destinationImage[destinationIndex++] = image[startIndex++];
00568                         if (pixelSize == 4)
```

```
00569                         {
00570                             destinationImage[destinationIndex++] = image[startIndex++];
00571                         }
00572                     }
00573                 }
00574             }
00575             else if (filter == FilterModes.Average)
00576             {
00577                 destinationImage[destinationIndex++] = 3;
00578                 for (int x = 0; x < width; x++)
00579                 {
00580                     if (x > 0 && y > 0)
00581                     {
00582                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00583                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00584                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00585                         if (pixelSize == 4)
00586                         {
00587                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00588                         }
00589                     }
00590                     else if (x > 0 && y == 0)
00591                     {
00592                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00593                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00594                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00595                         if (pixelSize == 4)
00596                         {
00597                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00598                         }
00599                     }
00600                     else if (x == 0 && y > 0)
00601                     {
00602                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00603                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00604                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00605                         if (pixelSize == 4)
00606                         {
00607                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00608                         }
00609                     }
00610                     else
00611                     {
00612                         destinationImage[destinationIndex++] = image[startIndex++];
00613                         destinationImage[destinationIndex++] = image[startIndex++];
00614                         destinationImage[destinationIndex++] = image[startIndex++];
00615                         if (pixelSize == 4)
00616                         {
00617                             destinationImage[destinationIndex++] = image[startIndex++];
00618                         }
00619                     }
00620                 }
00621             }
00622             else if (filter == FilterModes.Paeth)
00623             {
00624                 destinationImage[destinationIndex++] = 4;
00625                 for (int x = 0; x < width; x++)
00626                 {
00627                     if (x > 0 && y > 0)
00628                     {
00629                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00630                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00631                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00632                         if (pixelSize == 4)
00633                         {
00634                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00635                         }
```

```
00636                     }
00637                     else if (x > 0 && y == 0)
00638                     {
00639                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00640                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00641                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00642                         if (pixelSize == 4)
00643                         {
00644                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00645                         }
00646                     }
00647                     else if (x == 0 && y > 0)
00648                     {
00649                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00650                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00651                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00652                         if (pixelSize == 4)
00653                         {
00654                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00655                         }
00656                     }
00657                     else
00658                     {
00659                         destinationImage[destinationIndex++] = image[startIndex++];
00660                         destinationImage[destinationIndex++] = image[startIndex++];
00661                         destinationImage[destinationIndex++] = image[startIndex++];
00662                         if (pixelSize == 4)
00663                         {
00664                             destinationImage[destinationIndex++] = image[startIndex++];
00665                         }
00666                     }
00667                 }
00668             }
00669             else if (filter == FilterModes.Adaptive)
00670             {
00671                 int tempIndex = 0;
00672 
00673                 int none = 0;
00674                 int sub = 0;
00675                 int up = 0;
00676                 int average = 0;
00677                 int paeth = 0;
00678 
00679                 for (int x = 0; x < width; x++)
00680                 {
00681                     for (int i = 0; i < pixelSize; i++)
00682                     {
00683                         //none
00684                         tempBuffer[tempIndex] = image[startIndex];
00685 
00686                         //sub
00687                         if (x > 0)
00688                         {
00689                             tempBuffer[tempIndex + stride] = (byte)(image[startIndex] -
      image[startIndex - pixelSize]);
00690                         }
00691                         else
00692                         {
00693                             tempBuffer[tempIndex + stride] = image[startIndex];
00694                         }
00695 
00696                         //up
00697                         if (y > 0)
00698                         {
00699                             tempBuffer[tempIndex + stride * 2] = (byte)(image[startIndex] -
      image[startIndex - stride]);
00700                         }
00701                         else
00702                         {
00703                             tempBuffer[tempIndex + stride * 2] = image[startIndex];
00704                         }
00705 
00706                         //average
00707                         if (x > 0 && y > 0)
00708                         {
00709                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex - stride]) / 2);
00710                         }
00711                         else if (x > 0 && y == 0)
```

```
00712                         {
00713                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      image[startIndex - pixelSize] / 2);
00714                         }
00715                         else if (x == 0 && y > 0)
00716                         {
00717                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      image[startIndex - stride] / 2);
00718                         }
00719                         else
00720                         {
00721                             tempBuffer[tempIndex + stride * 3] = image[startIndex];
00722                         }
00723 
00724                         //paeth
00725                         if (x > 0 && y > 0)
00726                         {
00727                             tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex - pixelSize
      - stride]));
00728                         }
00729                         else if (x > 0 && y == 0)
00730                         {
00731                             tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      image[startIndex - pixelSize]);
00732                         }
00733                         else if (x == 0 && y > 0)
00734                         {
00735                             tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      image[startIndex - stride]);
00736                         }
00737                         else
00738                         {
00739                             tempBuffer[tempIndex + stride * 4] = image[startIndex];
00740                         }
00741 
00742                         none += tempBuffer[tempIndex];
00743                         sub += tempBuffer[tempIndex + stride];
00744                         up += tempBuffer[tempIndex + stride * 2];
00745                         average += tempBuffer[tempIndex + stride * 3];
00746                         paeth += tempBuffer[tempIndex + stride * 4];
00747                         tempIndex++;
00748                         startIndex++;
00749                     }
00750                 }
00751 
00752                 if (paeth <= average && paeth <= up && paeth <= sub && paeth <= none)
00753                 {
00754                     destinationImage[destinationIndex++] = 4;
00755                     tempIndex = stride * 4;
00756                     for (int i = 0; i < stride; i++)
00757                     {
00758                         destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00759                     }
00760                 }
00761                 else if (average <= up && average <= sub && average <= none)
00762                 {
00763                     destinationImage[destinationIndex++] = 3;
00764                     tempIndex = stride * 3;
00765                     for (int i = 0; i < stride; i++)
00766                     {
00767                         destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00768                     }
00769                 }
00770                 else if (up <= sub && up <= none)
00771                 {
00772                     destinationImage[destinationIndex++] = 2;
00773                     tempIndex = stride * 2;
00774                     for (int i = 0; i < stride; i++)
00775                     {
00776                         destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00777                     }
00778                 }
00779                 else if (sub <= none)
00780                 {
00781                     destinationImage[destinationIndex++] = 1;
00782                     tempIndex = stride;
00783                     for (int i = 0; i < stride; i++)
00784                     {
00785                         destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00786                     }
00787                 }
00788                 else
00789                 {
00790                     destinationImage[destinationIndex++] = 0;
00791                     tempIndex = 0;
00792                     for (int i = 0; i < stride; i++)
```

```
00793                     {
00794                         destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00795                     }
00796                 }
00797             }
00798         }
00799 
00800         internal unsafe static IntPtr FilterImageData(byte* image, int width, int height, int
      pixelSize, FilterModes filter)
00801         {
00802             IntPtr filteredMemory = Marshal.AllocHGlobal(height * (1 + width * pixelSize));
00803 
00804             IntPtr tempBuffer;
00805 
00806             if (filter == FilterModes.Adaptive)
00807             {
00808                 tempBuffer = Marshal.AllocHGlobal(5 * width * pixelSize);
00809             }
00810             else
00811             {
00812                 tempBuffer = IntPtr.Zero;
00813             }
00814 
00815             int stride = width * pixelSize;
00816 
00817             byte* filteredPointer = (byte*)filteredMemory;
00818 
00819             byte* tempPointer = (byte*)tempBuffer;
00820 
00821             for (int y = 0; y < height; y++)
00822             {
00823                 FilterRow(image, width, pixelSize, stride, y, filter, filteredPointer, tempPointer);
00824             }
00825 
00826             Marshal.FreeHGlobal(tempBuffer);
00827 
00828             return filteredMemory;
00829         }
00830 
00831         internal unsafe static IntPtr FilterImageData(byte* image, int width, int height, int
      pixelSize, FilterModes filter, int threadCount)
00832         {
00833             threadCount = Math.Min(threadCount, height);
00834 
00835             IntPtr filteredMemory = Marshal.AllocHGlobal(height * (1 + width * pixelSize));
00836 
00837             int stride = width * pixelSize;
00838 
00839             byte* filteredPointer = (byte*)filteredMemory;
00840 
00841             IntPtr[] tempBuffers = new IntPtr[threadCount];
00842 
00843             Thread[] threads = new Thread[threadCount];
00844             int[] rowsByThread = new int[threadCount];
00845             EventWaitHandle[] signalsFromThreads = new EventWaitHandle[threadCount];
00846             EventWaitHandle[] signalsToThreads = new EventWaitHandle[threadCount];
00847 
00848             int firstNeededRow = threadCount;
00849 
00850             for (int i = 0; i < threadCount; i++)
00851             {
00852                 rowsByThread[i] = i;
00853                 signalsFromThreads[i] = new EventWaitHandle(false, EventResetMode.ManualReset);
00854                 signalsToThreads[i] = new EventWaitHandle(false, EventResetMode.ManualReset);
00855 
00856                 if (filter == FilterModes.Adaptive)
00857                 {
00858                     tempBuffers[i] = Marshal.AllocHGlobal(5 * width * pixelSize);
00859                 }
00860                 else
00861                 {
00862                     tempBuffers[i] = IntPtr.Zero;
00863                 }
00864 
00865                 byte* tempPointer = (byte*)tempBuffers[i];
00866 
00867                 int threadIndex = i;
00868 
00869                 threads[i] = new Thread(() =>
00870                 {
00871                     while (rowsByThread[threadIndex] >= 0)
00872                     {
00873                         FilterRow(image, width, pixelSize, stride, rowsByThread[threadIndex], filter,
      filteredPointer, tempPointer);
00874                         EventWaitHandle.SignalAndWait(signalsFromThreads[threadIndex],
      signalsToThreads[threadIndex]);
00875                         signalsToThreads[threadIndex].Reset();
```

```
00876                     }
00877                 });
00878             }
00879
00880             int finished = 0;
00881
00882             for (int i = 0; i < threadCount; i++)
00883             {
00884                 threads[i].Start();
00885             }
00886
00887             while (finished < threadCount)
00888             {
00889                 int threadInd = EventWaitHandle.WaitAny(signalsFromThreads);
00890
00891                 signalsFromThreads[threadInd].Reset();
00892
00893                 if (firstNeededRow < height)
00894                 {
00895                     rowsByThread[threadInd] = firstNeededRow;
00896                     firstNeededRow++;
00897                     signalsToThreads[threadInd].Set();
00898                 }
00899                 else
00900                 {
00901                     rowsByThread[threadInd] = -1;
00902                     signalsToThreads[threadInd].Set();
00903                     finished++;
00904                 }
00905             }
00906
00907             for (int i = 0; i < threadCount; i++)
00908             {
00909                 if (filter == FilterModes.Adaptive)
00910                 {
00911                     Marshal.FreeHGlobal(tempBuffers[i]);
00912                 }
00913             }
00914
00915             return filteredMemory;
00916         }
00917
00918         //Based on http://www.libpng.org/pub/png/spec/1.2/PNG-Filters.html
00919         internal static byte PaethPredictor(byte a, byte b, byte c)
00920         {
00921             int p = (int)a + (int)b - (int)c;
00922             int pa = Math.Abs(p - (int)a);
00923             int pb = Math.Abs(p - (int)b);
00924             int pc = Math.Abs(p - (int)c);
00925
00926             if (pa <= pb && pa <= pc)
00927             {
00928                 return a;
00929             }
00930             else if (pb <= pc)
00931             {
00932                 return b;
00933             }
00934             else
00935             {
00936                 return c;
00937             }
00938         }
00939     }
00940
00941     //Derived from http://www.libpng.org/pub/png/spec/1.2/PNG-CRCAppendix.html
00942     internal static class CRC32
00943     {
00944         /* Table of CRCs of all 8-bit messages.  */
00945         private static readonly uint[] crc_table = new uint[256];
00946
00947         /* Flag:  has the table been computed?  Initially false.  */
00948         private static bool crc_table_computed = false;
00949
00950         /* Make the table for a fast CRC. */
00951         private static void MakeCRCtable()
00952         {
00953             uint c;
00954             int n, k;
00955
00956             for (n = 0; n < 256; n++)
00957             {
00958                 c = (uint)n;
00959                 for (k = 0; k < 8; k++)
00960                 {
00961                     if ((c & 1) != 0)
00962                     {
```

```
00963                       c = 0xedb88320 ^ (c >> 1);
00964                   }
00965                   else
00966                   {
00967                       c >>= 1;
00968                   }
00969               }
00970               crc_table[n] = c;
00971           }
00972           crc_table_computed = true;
00973       }
00974
00975       /* Update a running CRC with the bytes buf[0..len-1]--the CRC
00976 should be initialized to all 1's, and the transmitted value
00977 is the 1's complement of the final running CRC (see the
00978 crc() routine below)).  */
00979
00980       private static uint UpdateCRC(uint crc, Stream buf)
00981       {
00982           uint c = crc;
00983           int n;
00984
00985           if (!crc_table_computed)
00986           {
00987               MakeCRCtable();
00988           }
00989
00990           buf.Seek(0, SeekOrigin.Begin);
00991
00992           for (n = 0; n < buf.Length; n++)
00993           {
00994               c = crc_table[(c ^ (byte)buf.ReadByte()) & 0xff] ^ (c >> 8);
00995           }
00996           return c;
00997       }
00998
00999       private static uint UpdateCRC(uint crc, byte[] buf, int length)
01000       {
01001           uint c = crc;
01002           int n;
01003
01004           if (!crc_table_computed)
01005           {
01006               MakeCRCtable();
01007           }
01008
01009           for (n = 0; n < length; n++)
01010           {
01011               c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
01012           }
01013           return c;
01014       }
01015
01016       /* Return the CRC of the bytes buf[0..len-1].  */
01017       public static uint ComputeCRC(Stream buf)
01018       {
01019           return UpdateCRC(0xffffffff, buf) ^ 0xffffffff;
01020       }
01021
01022       public static uint ComputeCRC(byte[] buf, int length, byte[] prefix)
01023       {
01024           uint temp = UpdateCRC(0xffffffff, prefix, prefix.Length);
01025           return UpdateCRC(temp, buf, length) ^ 0xffffffff;
01026       }
01027   }
01028
01029   internal static class StreamUtils
01030   {
01031       public static void WriteASCIIString(this Stream sr, string val)
01032       {
01033           foreach (char c in val)
01034           {
01035               sr.WriteByte((byte)c);
01036           }
01037       }
01038
01039       internal static MemoryStream ZLibCompress(IntPtr memoryAddress, int memoryLength)
01040       {
01041           MemoryStream compressedStream = new MemoryStream();
01042           compressedStream.Write(new byte[] { 0x78, 0x01 }, 0, 2);
01043
01044           byte[] buffer = new byte[1024];
01045
01046           AdlerHolder adlerHolder = new AdlerHolder();
01047
01048           using (DeflateStream deflate = new DeflateStream(compressedStream,
      CompressionLevel.Optimal, true))
```

```
01049             {
01050                 while (memoryLength > 0)
01051                 {
01052                     int currentBytes = Math.Min(memoryLength, 1024);
01053                     Marshal.Copy(memoryAddress, buffer, 0, currentBytes);
01054                     deflate.Write(buffer, 0, currentBytes);
01055                     memoryLength -= currentBytes;
01056                     memoryAddress = IntPtr.Add(memoryAddress, currentBytes);
01057                     Adler32(buffer, currentBytes, adlerHolder);
01058                 }
01059             }
01060 
01061 
01062             uint checksum = (adlerHolder.s2 << 16) + adlerHolder.s1;
01063 
01064             compressedStream.Write(new byte[] { (byte)((checksum >> 24) & 255), (byte)((checksum >> 16)
      & 255), (byte)((checksum >> 8) & 255), (byte)(checksum & 255) }, 0, 4);
01065 
01066             compressedStream.Seek(0, SeekOrigin.Begin);
01067 
01068             return compressedStream;
01069         }
01070 
01071         internal static uint Adler32(Stream contentStream)
01072         {
01073             uint s1 = 1;
01074             uint s2 = 0;
01075 
01076             int readByte;
01077 
01078             while ((readByte = contentStream.ReadByte()) >= 0)
01079             {
01080                 s1 = (s1 + (byte)readByte) % 65521U;
01081                 s2 = (s2 + s1) % 65521U;
01082             }
01083 
01084             return (s2 << 16) + s1;
01085         }
01086 
01087 
01088         internal class AdlerHolder
01089         {
01090             public uint s1;
01091             public uint s2;
01092 
01093             public AdlerHolder()
01094             {
01095                 s1 = 1;
01096                 s2 = 0;
01097             }
01098         }
01099 
01100         internal static void Adler32(byte[] buffer, int length, AdlerHolder holder)
01101         {
01102             for (int i = 0; i < length; i++)
01103             {
01104                 holder.s1 = (holder.s1 + buffer[i]) % 65521U;
01105                 holder.s2 = (holder.s2 + holder.s1) % 65521U;
01106             }
01107         }
01108     }
01109 }
```

8.71 Point.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
```

```
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp
00024 {
00025 /// <summary>
00026 /// Represents a point relative to an origin in the top-left corner.
00027 /// </summary>
00028     public struct Point :  IReadOnlyList<double>
00029     {
00030 /// <summary>
00031 /// Horizontal (x) coordinate, measured to the right of the origin.
00032 /// </summary>
00033         public double X;
00034
00035 /// <summary>
00036 /// Vertical (y) coordinate, measured to the bottom of the origin.
00037 /// </summary>
00038         public double Y;
00039
00040 /// <inheritdoc/>
00041         public int Count => 2;
00042
00043 /// <inheritdoc/>
00044         public double this[int index]
00045         {
00046             get
00047             {
00048                 if (index == 0)
00049                 {
00050                     return this.X;
00051                 }
00052                 else if (index == 1)
00053                 {
00054                     return this.Y;
00055                 }
00056                 else
00057                 {
00058                     throw new IndexOutOfRangeException();
00059                 }
00060             }
00061         }
00062
00063 /// <summary>
00064 /// Create a new <see cref="Point"/>.
00065 /// </summary>
00066 /// <param name="x">The horizontal (x) coordinate.</param>
00067 /// <param name="y">The vertical (y) coordinate.</param>
00068         public Point(double x, double y)
00069         {
00070             X = x;
00071             Y = y;
00072         }
00073
00074 /// <summary>
00075 /// Computes the modulus of the vector represented by the <see cref="Point"/>.
00076 /// </summary>
00077 /// <returns>The modulus of the vector represented by the <see cref="Point"/>.</returns>
00078         public double Modulus()
00079         {
00080             return Math.Sqrt(X * X + Y * Y);
00081         }
00082
00083 /// <summary>
00084 /// Normalises a <see cref="Point"/>.
00085 /// </summary>
00086 /// <returns>The normalised <see cref="Point"/>.</returns>
00087         public Point Normalize()
00088         {
00089             double mod = Modulus();
00090             return new Point(X / mod, Y / mod);
00091         }
00092
00093 /// <summary>
00094 /// Checks whether this <see cref="Point"/> is equal to another <see cref="Point"/>, up to a specified
      tolerance.
00095 /// </summary>
00096 /// <param name="p2">The <see cref="Point"/> to compare.</param>
00097 /// <param name="tolerance">The tolerance threshold.</param>
00098 /// <returns><see langword="true"/> if both coordinates of the <see cref="Point"/>s are closer than
      <paramref name="tolerance"/> or if their relative difference (i.e.  <c>(a - b) / (a + b) * 2</c>) is
      smaller than <paramref name="tolerance"/>.  <see langword="false"/> otherwise.</returns>
00099         public bool IsEqual(Point p2, double tolerance)
00100         {
00101             return (Math.Abs(p2.X - this.X) <= tolerance || Math.Abs((p2.X - this.X) / (p2.X +
      this.X)) <= tolerance * 0.5) && (Math.Abs(p2.Y - this.Y) <= tolerance || Math.Abs((p2.Y - this.Y) /
```

```
      (p2.Y + this.Y)) <= tolerance * 0.5);
00102         }
00103
00104 /// <summary>
00105 /// Computes the top-left corner of the <see cref="Rectangle"/> identified by two <see
      cref="Point"/>s.
00106 /// </summary>
00107 /// <param name="p1">The first point.</param>
00108 /// <param name="p2">The second point.</param>
00109 /// <returns>A <see cref="Point"/> whose <see cref="X"/> coordinate is the smallest between the one of
      <paramref name="p1"/> and <paramref name="p2"/>, and likewise for the <see cref="Y"/>
      coordinate.</returns>
00110         public static Point Min(Point p1, Point p2)
00111         {
00112             return new Point(Math.Min(p1.X, p2.X), Math.Min(p1.Y, p2.Y));
00113         }
00114
00115 /// <summary>
00116 /// Computes the bottom-right corner of the <see cref="Rectangle"/> identified by two <see
      cref="Point"/>s.
00117 /// </summary>
00118 /// <param name="p1">The first point.</param>
00119 /// <param name="p2">The second point.</param>
00120 /// <returns>A <see cref="Point"/> whose <see cref="X"/> coordinate is the largest between the one of
      <paramref name="p1"/> and <paramref name="p2"/>, and likewise for the <see cref="Y"/>
      coordinate.</returns>
00121         public static Point Max(Point p1, Point p2)
00122         {
00123             return new Point(Math.Max(p1.X, p2.X), Math.Max(p1.Y, p2.Y));
00124         }
00125
00126 /// <summary>
00127 /// Computes the smallest <see cref="Rectangle"/> that contains all the specified points.
00128 /// </summary>
00129 /// <param name="points">The points whose bounds are being computed.</param>
00130 /// <returns>The smallest <see cref="Rectangle"/> that contains all the specified points.</returns>
00131         public static Rectangle Bounds(IEnumerable<Point> points)
00132         {
00133             bool initialised = false;
00134             Point min = new Point(double.NaN, double.NaN);
00135             Point max = new Point(double.NaN, double.NaN);
00136
00137             foreach (Point pt in points)
00138             {
00139                 if (!initialised)
00140                 {
00141                     min = pt;
00142                     max = pt;
00143                     initialised = true;
00144                 }
00145                 else
00146                 {
00147                     min = Point.Min(min, pt);
00148                     max = Point.Max(max, pt);
00149                 }
00150             }
00151
00152             return new Rectangle(min, max);
00153         }
00154
00155 /// <summary>
00156 /// Computes the smallest <see cref="Rectangle"/> that contains all the specified points.
00157 /// </summary>
00158 /// <param name="points">The points whose bounds are being computed.</param>
00159 /// <returns>The smallest <see cref="Rectangle"/> that contains all the specified points.</returns>
00160         public static Rectangle Bounds(params Point[] points)
00161         {
00162             return Bounds((IEnumerable<Point>)points);
00163         }
00164
00165 /// <inheritdoc/>
00166         public IEnumerator<double> GetEnumerator()
00167         {
00168             yield return this.X;
00169             yield return this.Y;
00170         }
00171
00172 /// <inheritdoc/>
00173         IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
00174
00175 /// <summary>
00176 /// Implicit conversion to a tuple.
00177 /// </summary>
00178 /// <param name="point">The point to convert.</param>
00179         public static implicit operator (double X, double Y)(Point point)
00180         {
00181             return (point.X, point.Y);
```

```
00182         }
00183
00184 /// <summary>
00185 /// Implicit conversion to an array.
00186 /// </summary>
00187 /// <param name="point">The point to convert</param>
00188         public static implicit operator double[](Point point)
00189         {
00190             return new double[] { point.X, point.Y };
00191         }
00192
00193 /// <summary>
00194 /// Implicit conversion from a tuple.
00195 /// </summary>
00196 /// <param name="tuple">The tuple to convert.</param>
00197         public static implicit operator Point((double X, double Y) tuple)
00198         {
00199             return new Point(tuple.X, tuple.Y);
00200         }
00201
00202 /// <summary>
00203 /// Explicit conversion from an array (will throw an exception if the array does not contain exactly
      two elements).
00204 /// </summary>
00205 /// <param name="array">The array to convert.  It must contain exactly two elements.</param>
00206         public static explicit operator Point(double[] array)
00207         {
00208             if (array == null)
00209             {
00210                 throw new ArgumentNullException("array", "The array to convert into a Point must not
      be null!");
00211             }
00212             else if (array.Length != 2)
00213             {
00214                 throw new ArgumentOutOfRangeException("array", array.Length.ToString(), "The length of
      the array to convert into a Point must be exactly 2!");
00215             }
00216
00217             return new Point(array[0], array[1]);
00218         }
00219
00220 /// <summary>
00221 /// Subtract the coordinates of two points.
00222 /// </summary>
00223 /// <param name="p1">The first point.</param>
00224 /// <param name="p2">The second point.</param>
00225 /// <returns>A new <see cref="Point"/> whose coordinates are the difference between the coordinates of
      the original points.</returns>
00226         public static Point operator -(Point p1, Point p2)
00227         {
00228             return new Point(p1.X - p2.X, p1.Y - p2.Y);
00229         }
00230
00231 /// <summary>
00232 /// Add the coordinates of two points.
00233 /// </summary>
00234 /// <param name="p1">The first point.</param>
00235 /// <param name="p2">The second point.</param>
00236 /// <returns>A new <see cref="Point"/> whose coordinates are the sum of the coordinates of the
      original points.</returns>
00237         public static Point operator +(Point p1, Point p2)
00238         {
00239             return new Point(p1.X + p2.X, p1.Y + p2.Y);
00240         }
00241
00242 /// <summary>
00243 /// Multiply the coordinates of a point by a scalar.
00244 /// </summary>
00245 /// <param name="t">The scalar.</param>
00246 /// <param name="p">The point.</param>
00247 /// <returns>A new <see cref="Point"/> whose coordinates are the product of the original points' and
      the scalar.</returns>
00248         public static Point operator *(double t, Point p)
00249         {
00250             return new Point(t * p.X, t * p.Y);
00251         }
00252
00253 /// <summary>
00254 /// Multiply the coordinates of a point by a scalar.
00255 /// </summary>
00256 /// <param name="t">The scalar.</param>
00257 /// <param name="p">The point.</param>
00258 /// <returns>A new <see cref="Point"/> whose coordinates are the product of the original points' and
      the scalar.</returns>
00259         public static Point operator *(Point p, double t)
00260         {
00261             return new Point(t * p.X, t * p.Y);
```

```
00262         }
00263     }
00264
00265 /// <summary>
00266 /// Represents the size of an object.
00267 /// </summary>
00268     public struct Size
00269     {
00270 /// <summary>
00271 /// Width of the object.
00272 /// </summary>
00273         public double Width;
00274
00275 /// <summary>
00276 /// Height of the object.
00277 /// </summary>
00278         public double Height;
00279
00280 /// <summary>
00281 /// Create a new <see cref="Size"/>.
00282 /// </summary>
00283 /// <param name="width">The width of the object.</param>
00284 /// <param name="height">The height of the object.</param>
00285         public Size(double width, double height)
00286         {
00287             Width = width;
00288             Height = height;
00289         }
00290     }
00291
00292 /// <summary>
00293 /// Represents a rectangle.
00294 /// </summary>
00295     public struct Rectangle
00296     {
00297 /// <summary>
00298 /// A rectangle whose dimensions are all <see cref="double.NaN"/>.
00299 /// </summary>
00300         public static readonly Rectangle NaN = new Rectangle(double.NaN, double.NaN, double.NaN,
      double.NaN);
00301
00302 /// <summary>
00303 /// Checks whether the rectangle is equivalent to <see cref="Rectangle.NaN"/>.
00304 /// </summary>
00305 /// <returns><see langword="true"/> if all the dimensions of the rectangle are <see
      cref="double.NaN"/>, <see langword="false"/> otherwise.</returns>
00306         public bool IsNaN()
00307         {
00308             return double.IsNaN(this.Location.X) && double.IsNaN(this.Location.Y) &&
      double.IsNaN(this.Size.Width) && double.IsNaN(this.Size.Height);
00309         }
00310
00311 /// <summary>
00312 /// The top-left corner of the rectangle.
00313 /// </summary>
00314         public Point Location;
00315
00316 /// <summary>
00317 /// The size of the rectangle.
00318 /// </summary>
00319         public Size Size;
00320
00321 /// <summary>
00322 /// The centre of the rectangle.
00323 /// </summary>
00324         public Point Centre { get { return new Point(this.Location.X + this.Size.Width * 0.5,
      this.Location.Y + this.Size.Height * 0.5); } }
00325
00326 /// <summary>
00327 /// Create a new <see cref="Rectangle"/> given its top-left corner and its size.
00328 /// </summary>
00329 /// <param name="location">The top-left corner of the rectangle.</param>
00330 /// <param name="size">The size of the rectangle.</param>
00331         public Rectangle(Point location, Size size)
00332         {
00333             this.Location = location;
00334             this.Size = size;
00335         }
00336
00337 /// <summary>
00338 /// Create a new <see cref="Rectangle"/> given its top-left corner and its size.
00339 /// </summary>
00340 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle.</param>
00341 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle.</param>
00342 /// <param name="width">The width of the rectangle.</param>
00343 /// <param name="height">The height of the rectangle.</param>
00344         public Rectangle(double x, double y, double width, double height)
```

```
00345         {
00346             Location = new Point(x, y);
00347             Size = new Size(width, height);
00348         }
00349 
00350 /// <summary>
00351 /// Create a new <see cref="Rectangle"/> given its top-left corner and its bottom-right corner.
00352 /// </summary>
00353 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00354 /// <param name="bottomRight">The bottom-right corner of the rectangle.</param>
00355         public Rectangle(Point topLeft, Point bottomRight)
00356         {
00357             this.Location = topLeft;
00358             this.Size = new Size(bottomRight.X - topLeft.X, bottomRight.Y - topLeft.Y);
00359         }
00360 
00361 /// <summary>
00362 /// Computes the rectangular bounds of the union of two <see cref="Rectangle"/>s.
00363 /// </summary>
00364 /// <param name="rectangle1">The first <see cref="Rectangle"/>.</param>
00365 /// <param name="rectangle2">The second <see cref="Rectangle"/>.</param>
00366 /// <returns>The smallest <see cref="Rectangle"/> containing both <paramref name="rectangle1"/> and
      <paramref name="rectangle2"/>.</returns>
00367         public static Rectangle Union(Rectangle rectangle1, Rectangle rectangle2)
00368         {
00369             double minX = rectangle1.Location.X;
00370             double minY = rectangle1.Location.Y;
00371             double maxX = rectangle1.Location.X;
00372             double maxY = rectangle1.Location.Y;
00373 
00374             minX = Math.Min(minX, rectangle1.Location.X + rectangle1.Size.Width);
00375             minX = Math.Min(minX, rectangle2.Location.X);
00376             minX = Math.Min(minX, rectangle2.Location.X + rectangle2.Size.Width);
00377 
00378             minY = Math.Min(minY, rectangle1.Location.Y + rectangle1.Size.Height);
00379             minY = Math.Min(minY, rectangle2.Location.Y);
00380             minY = Math.Min(minY, rectangle2.Location.Y + rectangle2.Size.Height);
00381 
00382 
00383             maxX = Math.Max(maxX, rectangle1.Location.X + rectangle1.Size.Width);
00384             maxX = Math.Max(maxX, rectangle2.Location.X);
00385             maxX = Math.Max(maxX, rectangle2.Location.X + rectangle2.Size.Width);
00386 
00387             maxY = Math.Max(maxY, rectangle1.Location.Y + rectangle1.Size.Height);
00388             maxY = Math.Max(maxY, rectangle2.Location.Y);
00389             maxY = Math.Max(maxY, rectangle2.Location.Y + rectangle2.Size.Height);
00390 
00391             return new Rectangle(minX, minY, maxX - minX, maxY - minY);
00392         }
00393 
00394 /// <summary>
00395 /// Computes the intersection of two <see cref="Rectangle"/>s.
00396 /// </summary>
00397 /// <param name="rectangle1">The first <see cref="Rectangle"/>.</param>
00398 /// <param name="rectangle2">The second <see cref="Rectangle"/>.</param>
00399 /// <returns>The rectangle corresponding to the intersection of <paramref name="rectangle1"/> and
      <paramref name="rectangle2"/>, or <see cref="Rectangle.NaN"/> if the intersection is empty.</returns>
00400         public static Rectangle Intersection(Rectangle rectangle1, Rectangle rectangle2)
00401         {
00402             double x0 = Math.Max(rectangle1.Location.X, rectangle2.Location.X);
00403             double x1 = Math.Min(rectangle1.Location.X + rectangle1.Size.Width, rectangle2.Location.X
      + rectangle2.Size.Width);
00404 
00405             double y0 = Math.Max(rectangle1.Location.Y, rectangle2.Location.Y);
00406             double y1 = Math.Min(rectangle1.Location.Y + rectangle1.Size.Height, rectangle2.Location.Y
      + rectangle2.Size.Height);
00407 
00408             if (x1 >= x0 && y1 >= y0)
00409             {
00410                 return new Rectangle(x0, y0, x1 - x0, y1 - y0);
00411             }
00412             else
00413             {
00414                 return Rectangle.NaN;
00415             }
00416 
00417         }
00418 
00419 /// <summary>
00420 /// Computes the rectangular bounds of the union of multiple <see cref="Rectangle"/>s.
00421 /// </summary>
00422 /// <param name="rectangles">The <see cref="Rectangle"/>s whose union will be computed.</param>
00423 /// <returns>The smallest <see cref="Rectangle"/> containing all the <paramref
      name="rectangles"/>.</returns>
00424         public static Rectangle Union(IEnumerable<Rectangle> rectangles)
00425         {
00426             if (rectangles.Any())
```

```
00427             {
00428                 bool initialised = false;
00429 
00430                 Rectangle tbr = new Rectangle();
00431 
00432                 foreach (Rectangle rect in rectangles)
00433                 {
00434                     if (!initialised)
00435                     {
00436                         tbr = rect;
00437                     }
00438                     else
00439                     {
00440                         tbr = Union(rect, tbr);
00441                     }
00442                 }
00443 
00444                 return tbr;
00445             }
00446             else
00447             {
00448                 return Rectangle.NaN;
00449             }
00450         }
00451 
00452 /// <summary>
00453 /// Computes the rectangular bounds of the union of multiple <see cref="Rectangle"/>s.
00454 /// </summary>
00455 /// <param name="rectangles">The <see cref="Rectangle"/>s whose union will be computed.</param>
00456 /// <returns>The smallest <see cref="Rectangle"/> containing all the <paramref
      name="rectangles"/>.</returns>
00457         public static Rectangle Union(params Rectangle[] rectangles)
00458         {
00459             return Union((IEnumerable<Rectangle>)rectangles);
00460         }
00461     }
00462 }
```

8.72 RasterImage.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.IO;
00020 using System.Runtime.InteropServices;
00021 
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents the pixel format of a raster image.
00026 /// </summary>
00027     public enum PixelFormats
00028     {
00029 /// <summary>
00030 /// RGB 24bpp format.
00031 /// </summary>
00032         RGB,
00033 
00034 /// <summary>
00035 /// RGBA 32bpp format.
00036 /// </summary>
00037         RGBA,
00038 
00039 /// <summary>
00040 /// BGR 24bpp format.
00041 /// </summary>
00042         BGR,
00043 
```

```
00044 /// <summary>
00045 /// BGR 32bpp format.
00046 /// </summary>
00047         BGRA
00048     }
00049
00050 /// <summary>
00051 /// An <see cref="IDisposable"/> wrapper around an <see cref="IntPtr"/> that frees the allocated
      memory when it is disposed.
00052 /// </summary>
00053     public class DisposableIntPtr :  IDisposable
00054     {
00055 /// <summary>
00056 /// The pointer to the unmanaged memory.
00057 /// </summary>
00058         public readonly IntPtr InternalPointer;
00059
00060 /// <summary>
00061 /// Create a new DisposableIntPtr.
00062 /// </summary>
00063 /// <param name="pointer">The pointer that should be freed upon disposing of this object.</param>
00064         public DisposableIntPtr(IntPtr pointer)
00065         {
00066             this.InternalPointer = pointer;
00067         }
00068
00069         private bool disposedValue;
00070
00071 ///<inheritdoc/>
00072         protected virtual void Dispose(bool disposing)
00073         {
00074             if (!disposedValue)
00075             {
00076                 Marshal.FreeHGlobal(InternalPointer);
00077                 disposedValue = true;
00078             }
00079         }
00080
00081 ///<inheritdoc/>
00082         ~DisposableIntPtr()
00083         {
00084             Dispose(disposing:  false);
00085         }
00086
00087 ///<inheritdoc/>
00088         public void Dispose()
00089         {
00090             Dispose(disposing:  true);
00091             GC.SuppressFinalize(this);
00092         }
00093     }
00094
00095 /// <summary>
00096 /// Represents a raster image, created from raw pixel data.  Consider using the derived classes
      included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or
      a <see cref="Stream"/>.
00097 /// </summary>
00098     public class RasterImage :  IDisposable
00099     {
00100 /// <summary>
00101 /// The memory address of the image pixel data.
00102 /// </summary>
00103         public IntPtr ImageDataAddress { get; protected set; }
00104
00105 /// <summary>
00106 /// An <see cref="IDisposable"/> that will be disposed when the image is disposed.
00107 /// </summary>
00108         public IDisposable DataHolder { get; protected set; }
00109
00110 /// <summary>
00111 /// A univocal identifier for this image.
00112 /// </summary>
00113         public string Id { get; protected set; }
00114
00115 /// <summary>
00116 /// Determines whether the image has an alpha channel.
00117 /// </summary>
00118         public bool HasAlpha { get; protected set; }
00119
00120 /// <summary>
00121 /// The width in pixels of the image.
00122 /// </summary>
00123         public int Width { get; protected set; }
00124
00125 /// <summary>
00126 /// The height in pixels of the image.
00127 /// </summary>
```

```
00128         public int Height { get; protected set; }
00129
00130 /// <summary>
00131 /// Determines whether the image should be interpolated when it is resized.
00132 /// </summary>
00133         public bool Interpolate { get; protected set; }
00134
00135         private MemoryStream _PNGStream = null;
00136
00137 /// <summary>
00138 /// Contains a representation of the image in PNG format.  Generated at the first access and cached
      until the image is disposed.
00139 /// </summary>
00140         public MemoryStream PNGStream
00141         {
00142             get
00143             {
00144                 if (_PNGStream == null)
00145                 {
00146                     _PNGStream = new MemoryStream();
00147                     this.EncodeAsPNG(_PNGStream);
00148                 }
00149                 _PNGStream.Seek(0, SeekOrigin.Begin);
00150                 return _PNGStream;
00151             }
00152         }
00153
00154 /// <summary>
00155 /// Default constructor, necessary for inheritance.
00156 /// </summary>
00157         protected RasterImage()
00158         {
00159
00160         }
00161
00162 /// <summary>
00163 /// Creates a new <see cref="RasterImage"/> instance from the specified pixel data in RGB or RGBA
      format.
00164 /// </summary>
00165 /// <param name="pixelData">The address of the image pixel data in RGB or RGBA format.</param>
00166 /// <param name="width">The width in pixels of the image.</param>
00167 /// <param name="height">The height in pixels of the image.</param>
00168 /// <param name="hasAlpha">true if the image is in RGBA format, false if it is in RGB format.</param>
00169 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00170         public RasterImage(IntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)
00171         {
00172             this.Id = Guid.NewGuid().ToString();
00173             this.ImageDataAddress = pixelData;
00174             this.Width = width;
00175             this.Height = height;
00176             this.HasAlpha = hasAlpha;
00177             this.Interpolate = interpolate;
00178         }
00179
00180 /// <summary>
00181 /// Creates a new <see cref="RasterImage"/> instance from the specified pixel data in RGB or RGBA
      format.
00182 /// </summary>
00183 /// <param name="pixelData">The address of the image pixel data in RGB or RGBA format wrapped in a
      <see cref="DisposableIntPtr"/>.  The <see cref="RasterImage"/> will take ownership of this
      memory.</param>
00184 /// <param name="width">The width in pixels of the image.</param>
00185 /// <param name="height">The height in pixels of the image.</param>
00186 /// <param name="hasAlpha">true if the image is in RGBA format, false if it is in RGB format.</param>
00187 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00188         public RasterImage(ref DisposableIntPtr pixelData, int width, int height, bool hasAlpha, bool
      interpolate)
00189         {
00190             this.Id = Guid.NewGuid().ToString();
00191             this.ImageDataAddress = pixelData.InternalPointer;
00192             this.DataHolder = pixelData;
00193             this.Width = width;
00194             this.Height = height;
00195             this.HasAlpha = hasAlpha;
00196             this.Interpolate = interpolate;
00197         }
00198
00199 /// <summary>
00200 /// Creates a new <see cref="RasterImage"/> instance copying the specified pixel data.
00201 /// </summary>
00202 /// <param name="data">The image pixel data that will be copied.</param>
00203 /// <param name="width">The width in pixels of the image.</param>
00204 /// <param name="height">The height in pixels of the image.</param>
00205 /// <param name="pixelFormat">The format of the pixel data.</param>
00206 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00207         public RasterImage(byte[] data, int width, int height, PixelFormats pixelFormat, bool
      interpolate)
```

```
00208         {
00209             this.ImageDataAddress = Marshal.AllocHGlobal(data.Length);
00210             GC.AddMemoryPressure(data.Length);
00211             this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00212             this.Id = Guid.NewGuid().ToString();
00213             this.Width = width;
00214             this.Height = height;
00215             this.Interpolate = interpolate;
00216 
00217             switch (pixelFormat)
00218             {
00219                 case PixelFormats.RGB:
00220                 case PixelFormats.RGBA:
00221                     Marshal.Copy(data, 0, this.ImageDataAddress, data.Length);
00222                     this.HasAlpha = pixelFormat == PixelFormats.RGBA;
00223                     break;
00224 
00225                 case PixelFormats.BGRA:
00226                 case PixelFormats.BGR:
00227                     this.HasAlpha = pixelFormat == PixelFormats.BGRA;
00228 
00229                     int pixelSize = pixelFormat == PixelFormats.BGRA ? 4 :  3;
00230                     int pixelCount = width * height;
00231 
00232                     unsafe
00233                     {
00234                         byte* dataPointer = (byte*)this.ImageDataAddress;
00235                         for (int i = 0; i < pixelCount; i++)
00236                         {
00237                             dataPointer[i * pixelSize] = data[i * pixelSize + 2];
00238                             dataPointer[i * pixelSize + 1] = data[i * pixelSize + 1];
00239                             dataPointer[i * pixelSize + 2] = data[i * pixelSize];
00240                             if (pixelSize == 4)
00241                             {
00242                                 dataPointer[i * pixelSize + 3] = data[i * pixelSize + 3];
00243                             }
00244                         }
00245                     }
00246                     break;
00247             }
00248         }
00249 
00250         private void EncodeAsPNG(Stream outputStream)
00251         {
00252             unsafe
00253             {
00254                 PNGEncoder.SavePNG((byte*)this.ImageDataAddress, this.Width, this.Height,
      this.HasAlpha, outputStream, PNGEncoder.FilterModes.Adaptive);
00255             }
00256         }
00257 
00258 /// <summary>
00259 /// Disposes the <see cref="PNGStream"/>.  Also useful if is is necessary to regenerate it, e.g.
      because the underlying image pixel data has changed.
00260 /// </summary>
00261         public void ClearPNGCache()
00262         {
00263             _PNGStream?.Dispose();
00264             _PNGStream = null;
00265         }
00266 
00267         private bool disposedValue;
00268 
00269 /// <inheritdoc/>
00270         protected virtual void Dispose(bool disposing)
00271         {
00272             if (!disposedValue)
00273             {
00274                 if (disposing)
00275                 {
00276                     _PNGStream?.Dispose();
00277 
00278                     if (DataHolder != null)
00279                     {
00280                         DataHolder.Dispose();
00281                         GC.RemoveMemoryPressure(Height * Width * (HasAlpha ?  4 :  3));
00282                     }
00283                 }
00284                 disposedValue = true;
00285             }
00286         }
00287 
00288 ///<inheritdoc/>
00289         ~RasterImage()
00290         {
00291             Dispose(disposing:  false);
00292         }
```

```
00293
00294 /// <inheritdoc/>
00295         public void Dispose()
00296         {
00297             Dispose(disposing:  true);
00298             GC.SuppressFinalize(this);
00299         }
00300     }
00301 }
```

8.73 Segment.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp
00023 {
00024
00025 /// <summary>
00026 /// Represents a segment as part of a <see cref="GraphicsPath"/>.
00027 /// </summary>
00028     public abstract class Segment
00029     {
00030
00031 /// <summary>
00032 /// The type of the <see cref="Segment"/>.
00033 /// </summary>
00034         public abstract SegmentType Type { get; }
00035
00036 /// <summary>
00037 /// The points used to define the <see cref="Segment"/>.
00038 /// </summary>
00039         public Point[] Points { get; protected set; }
00040
00041 /// <summary>
00042 /// The end point of the <see cref="Segment"/>.
00043 /// </summary>
00044         public virtual Point Point
00045         {
00046             get
00047             {
00048                 return Points[Points.Length - 1];
00049             }
00050         }
00051
00052 /// <summary>
00053 /// Creates a copy of the <see cref="Segment"/>.
00054 /// </summary>
00055 /// <returns>A copy of the <see cref="Segment"/>.</returns>
00056         public abstract Segment Clone();
00057
00058 /// <summary>
00059 /// Computes the length of the <see cref="Segment"/>.
00060 /// </summary>
00061 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00062 /// <returns>The length of the segment.</returns>
00063         public abstract double Measure(Point previousPoint);
00064
00065 /// <summary>
00066 /// Gets the point on the <see cref="Segment"/> at the specified (relative) <paramref
      name="position"/>).
00067 /// </summary>
00068 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
```

```
00069 /// <param name="position">The relative position on the <see cref="Segment"/> (0 is the start of the
      <see cref="Segment"/>, 1 is the end of the <see cref="Segment"/>).</param>
00070 /// <returns>The point at the specified position.</returns>
00071         public abstract Point GetPointAt(Point previousPoint, double position);
00072
00073 /// <summary>
00074 /// Gets the tangent to the <see cref="Segment"/> at the specified (relative) <paramref
      name="position"/>).
00075 /// </summary>
00076 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00077 /// <param name="position">The relative position on the <see cref="Segment"/> (0 is the start of the
      <see cref="Segment"/>, 1 is the end of the <see cref="Segment"/>).</param>
00078 /// <returns>The tangent to the point at the specified position.</returns>
00079         public abstract Point GetTangentAt(Point previousPoint, double position);
00080
00081 /// <summary>
00082 /// Transform the segment into a series of linear segments.  Segments that are already linear are not
      changed.
00083 /// </summary>
00084 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00085 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
00086 /// <returns>A collection of linear segments that approximate the current segment.</returns>
00087         public abstract IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution);
00088
00089 /// <summary>
00090 /// Gets the tanget at the points at which the segment would be linearised.
00091 /// </summary>
00092 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00093 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
00094 /// <returns>A collection of tangents at the points in which the segment would be
      linearised.</returns>
00095         public abstract IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution);
00096
00097 /// <summary>
00098 /// Applies an arbitrary transformation to all of the points of the <see cref="Segment"/>.
00099 /// </summary>
00100 /// <param name="transformationFunction">An arbitrary transformation function.</param>
00101 /// <returns>A collection of <see cref="Segment"/>s that have been transformed according to the
      <paramref name="transformationFunction"/>.</returns>
00102         public abstract IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction);
00103
00104 /// <summary>
00105 /// Flattens the <see cref="Segment"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path.
00106 /// </summary>
00107 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00108 /// <param name="flatness">The maximum deviation from the original path.</param>
00109 /// <returns>A collection of <see cref="Segment"/>s composed only of linear segments that approximates
      the current <see cref="Segment"/>.</returns>
00110         public abstract IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness);
00111
00112 /// <summary>
00113 /// Flattens the <see cref="Segment"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path, assuming that the
      <see cref="Segment"/> will be drawn with the specified <paramref name="offset"/>.
00114 /// </summary>
00115 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00116 /// <param name="offset">The offset that will be used to draw the <see cref="Segment"/> (e.g., the
      line width of the stroke).</param>
00117 /// <param name="flatness">The maximum deviation from the original path.</param>
00118 /// <returns>A collection of tuples where the first element is a <see cref="Point"/> representing the
      end-point of a linear segment, and the second element is a <see cref="Point"/> containing the value of
      the tangent to the original <see cref="Segment"/> at that point.</returns>
00119         public abstract IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness);
00120     }
00121
00122     internal class MoveSegment :  Segment
00123     {
00124         public override SegmentType Type => SegmentType.Move;
00125
00126         public MoveSegment(Point p)
00127         {
00128             this.Points = new Point[] { p };
00129         }
00130
00131         public MoveSegment(double x, double y)
00132         {
00133             this.Points = new Point[] { new Point(x, y) };
00134         }
00135
```

```
00136         public override Segment Clone()
00137         {
00138             return new MoveSegment(this.Point);
00139         }
00140 
00141         public override double Measure(Point previousPoint)
00142         {
00143             return 0;
00144         }
00145 
00146         public override Point GetPointAt(Point previousPoint, double position)
00147         {
00148             throw new InvalidOperationException();
00149         }
00150 
00151         public override Point GetTangentAt(Point previousPoint, double position)
00152         {
00153             throw new InvalidOperationException();
00154         }
00155 
00156         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00157         {
00158             yield return new MoveSegment(this.Point);
00159         }
00160 
00161         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00162         {
00163             throw new InvalidOperationException();
00164         }
00165 
00166         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00167         {
00168             yield return new MoveSegment(transformationFunction(this.Point));
00169         }
00170 
00171         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00172         {
00173             yield return new MoveSegment(this.Point);
00174         }
00175 
00176         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00177         {
00178             throw new InvalidOperationException();
00179         }
00180     }
00181 
00182     internal class LineSegment :  Segment
00183     {
00184         public override SegmentType Type => SegmentType.Line;
00185 
00186         public LineSegment(Point p)
00187         {
00188             this.Points = new Point[] { p };
00189         }
00190 
00191         public LineSegment(double x, double y)
00192         {
00193             this.Points = new Point[] { new Point(x, y) };
00194         }
00195 
00196         public override Segment Clone()
00197         {
00198             return new LineSegment(this.Point);
00199         }
00200 
00201         private double cachedLength = double.NaN;
00202 
00203         public override double Measure(Point previousPoint)
00204         {
00205             if (double.IsNaN(cachedLength))
00206             {
00207                 cachedLength = Math.Sqrt((this.Point.X - previousPoint.X) * (this.Point.X -
      previousPoint.X) + (this.Point.Y - previousPoint.Y) * (this.Point.Y - previousPoint.Y));
00208             }
00209 
00210             return cachedLength;
00211         }
00212 
00213         public override Point GetPointAt(Point previousPoint, double position)
00214         {
00215             return new Point(previousPoint.X * (1 - position) + this.Point.X * position,
      previousPoint.Y * (1 - position) + this.Point.Y * position);
00216         }
00217 
00218         public override Point GetTangentAt(Point previousPoint, double position)
```

```
00219         {
00220             return new Point(this.Point.X - previousPoint.X, this.Point.Y -
      previousPoint.Y).Normalize();
00221         }
00222 
00223         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00224         {
00225             yield return new LineSegment(this.Point);
00226         }
00227 
00228         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00229         {
00230             yield return this.GetTangentAt(previousPoint.Value, 1);
00231         }
00232 
00233         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00234         {
00235             yield return new LineSegment(transformationFunction(this.Point));
00236         }
00237 
00238         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00239         {
00240             yield return new LineSegment(this.Point);
00241         }
00242 
00243         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00244         {
00245             if (previousPoint != null)
00246             {
00247                 yield return (previousPoint.Value, this.GetTangentAt(previousPoint.Value, 1));
00248             }
00249 
00250             yield return (this.Point, this.GetTangentAt(previousPoint.Value, 1));
00251         }
00252     }
00253 
00254     internal class CloseSegment :  Segment
00255     {
00256         public override SegmentType Type => SegmentType.Close;
00257 
00258         public CloseSegment() { }
00259 
00260         public override Segment Clone()
00261         {
00262             return new CloseSegment();
00263         }
00264 
00265         public override double Measure(Point previousPoint)
00266         {
00267             return 0;
00268         }
00269 
00270         public override Point GetPointAt(Point previousPoint, double position)
00271         {
00272             throw new InvalidOperationException();
00273         }
00274 
00275         public override Point GetTangentAt(Point previousPoint, double position)
00276         {
00277             throw new InvalidOperationException();
00278         }
00279 
00280         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00281         {
00282             yield return new CloseSegment();
00283         }
00284 
00285         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00286         {
00287             throw new InvalidOperationException();
00288         }
00289 
00290         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00291         {
00292             yield return new CloseSegment();
00293         }
00294 
00295         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00296         {
00297             yield return new CloseSegment();
00298         }
00299 
00300         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
```

```
00301         {
00302             throw new InvalidOperationException();
00303         }
00304     }
00305
00306     internal class CubicBezierSegment :  Segment
00307     {
00308         public override SegmentType Type => SegmentType.CubicBezier;
00309         public CubicBezierSegment(double x1, double y1, double x2, double y2, double x3, double y3)
00310         {
00311             Points = new Point[] { new Point(x1, y1), new Point(x2, y2), new Point(x3, y3) };
00312         }
00313
00314         public CubicBezierSegment(Point p1, Point p2, Point p3)
00315         {
00316             Points = new Point[] { p1, p2, p3 };
00317         }
00318
00319         public override Segment Clone()
00320         {
00321             return new CubicBezierSegment(Points[0], Points[1], Points[2]);
00322         }
00323
00324         private double cachedLength = double.NaN;
00325         private int cachedSegments = -1;
00326
00327         public override double Measure(Point previousPoint)
00328         {
00329             if (double.IsNaN(cachedLength))
00330             {
00331                 int segments = 16;
00332                 double prevLength = 0;
00333                 double currLength = Measure(previousPoint, segments);
00334
00335                 while (currLength > 0.00001 && Math.Abs(currLength - prevLength) / currLength >
    0.0001)
00336                 {
00337                     segments *= 2;
00338                     prevLength = currLength;
00339                     currLength = Measure(previousPoint, segments);
00340                 }
00341
00342                 cachedSegments = segments;
00343
00344                 cachedLength = currLength;
00345             }
00346
00347             return cachedLength;
00348         }
00349
00350         public Point GetBezierPointAt(Point previousPoint, double position)
00351         {
00352             if (position <= 1 && position >= 0)
00353             {
00354                 return new Point(
00355                 this.Points[2].X * position * position * position + 3 * this.Points[1].X * position *
    position * (1 - position) + 3 * this.Points[0].X * position * (1 - position) * (1 - position) +
    previousPoint.X * (1 - position) * (1 - position) * (1 - position),
00356                 this.Points[2].Y * position * position * position + 3 * this.Points[1].Y * position *
    position * (1 - position) + 3 * this.Points[0].Y * position * (1 - position) * (1 - position) +
    previousPoint.Y * (1 - position) * (1 - position) * (1 - position)
00357                 );
00358             }
00359             else if (position > 1)
00360             {
00361                 Point tangent = GetBezierTangentAt(previousPoint, 1);
00362
00363                 double excessLength = (position - 1) * this.Measure(previousPoint);
00364
00365                 return new Point(this.Point.X + tangent.X * excessLength, this.Point.Y + tangent.Y *
    excessLength);
00366             }
00367             else
00368             {
00369                 Point tangent = GetBezierTangentAt(previousPoint, 0);
00370
00371                 return new Point(previousPoint.X + tangent.X * position * this.Measure(previousPoint),
    previousPoint.Y + tangent.Y * position * this.Measure(previousPoint));
00372             }
00373         }
00374
00375         public override Point GetPointAt(Point previousPoint, double position)
00376         {
00377             double t = GetTFromPosition(previousPoint, position);
00378             return this.GetBezierPointAt(previousPoint, t);
00379         }
00380
```

```
00381         public override Point GetTangentAt(Point previousPoint, double position)
00382         {
00383             double t = GetTFromPosition(previousPoint, position);
00384             return this.GetBezierTangentAt(previousPoint, t);
00385         }
00386 
00387         private double Measure(Point startPoint, int segments)
00388         {
00389             double delta = 1.0 / segments;
00390 
00391             double tbr = 0;
00392 
00393             for (int i = 1; i < segments; i++)
00394             {
00395                 Point p1 = GetBezierPointAt(startPoint, delta * (i - 1));
00396                 Point p2 = GetBezierPointAt(startPoint, delta * i);
00397 
00398                 tbr += Math.Sqrt((p1.X - p2.X) * (p1.X - p2.X) + (p1.Y - p2.Y) * (p1.Y - p2.Y));
00399             }
00400 
00401             return tbr;
00402         }
00403 
00404         private double Measure(Point startPoint, int segments, double maxT)
00405         {
00406             double delta = maxT / segments;
00407 
00408             double tbr = 0;
00409 
00410             for (int i = 1; i < segments; i++)
00411             {
00412                 Point p1 = GetBezierPointAt(startPoint, delta * (i - 1));
00413                 Point p2 = GetBezierPointAt(startPoint, delta * i);
00414 
00415                 tbr += Math.Sqrt((p1.X - p2.X) * (p1.X - p2.X) + (p1.Y - p2.Y) * (p1.Y - p2.Y));
00416             }
00417 
00418             return tbr;
00419         }
00420 
00421         private Point GetBezierTangentAt(Point previousPoint, double position)
00422         {
00423             if (position <= 1 && position >= 0)
00424             {
00425                 if (position == 0 && previousPoint.IsEqual(this.Points[0], GraphicsPath.Tolerance))
00426                 {
00427                     return (this.Points[1] - previousPoint).Normalize();
00428                 }
00429                 else if (position == 1 && this.Points[2].IsEqual(this.Points[1],
      GraphicsPath.Tolerance))
00430                 {
00431                     return (this.Points[2] - this.Points[0]).Normalize();
00432                 }
00433                 else
00434                 {
00435                     return new Point(
00436                         3 * this.Points[2].X * position * position +
00437                         3 * this.Points[1].X * position * (2 - 3 * position) +
00438                         3 * this.Points[0].X * (3 * position * position - 4 * position + 1) +
00439                         -3 * previousPoint.X * (1 - position) * (1 - position),
00440 
00441                         3 * this.Points[2].Y * position * position +
00442                         3 * this.Points[1].Y * position * (2 - 3 * position) +
00443                         3 * this.Points[0].Y * (3 * position * position - 4 * position + 1) +
00444                         -3 * previousPoint.Y * (1 - position) * (1 - position)).Normalize();
00445                 }
00446             }
00447             else if (position > 1)
00448             {
00449                 return GetBezierTangentAt(previousPoint, 1);
00450             }
00451             else
00452             {
00453                 return GetBezierTangentAt(previousPoint, 0);
00454             }
00455         }
00456 
00457         private double GetTFromPosition(Point previousPoint, double position)
00458         {
00459             if (position <= 0 || position >= 1)
00460             {
00461                 return position;
00462             }
00463             else
00464             {
00465                 double length = this.Measure(previousPoint);
00466 
```

```
00467                 double lowerBound = 0;
00468                 double upperBound = 0.5;
00469 
00470                 double lowerPos = 0;
00471                 double upperPos = Measure(previousPoint, (int)Math.Ceiling(this.cachedSegments *
      upperBound), upperBound) / length;
00472 
00473                 if (upperPos < position)
00474                 {
00475                     lowerBound = upperBound;
00476                     lowerPos = upperPos;
00477 
00478                     upperBound = 1;
00479                     upperPos = 1;
00480                 }
00481 
00482                 while (Math.Min(upperPos - position, position - lowerPos) > 0.001)
00483                 {
00484                     double mid = (lowerBound + upperBound) * 0.5;
00485                     double midPos = Measure(previousPoint, (int)Math.Ceiling(this.cachedSegments *
      mid), mid) / length;
00486 
00487                     if (midPos > position)
00488                     {
00489                         upperBound = mid;
00490                         upperPos = midPos;
00491                     }
00492                     else
00493                     {
00494                         lowerBound = mid;
00495                         lowerPos = midPos;
00496                     }
00497                 }
00498 
00499                 return lowerBound + (position - lowerPos) / (upperPos - lowerPos) * (upperBound -
      lowerBound);
00500             }
00501         }
00502 
00503         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00504         {
00505             double length = this.Measure(previousPoint.Value);
00506             int segmentCount = (int)Math.Ceiling(length / resolution);
00507 
00508             for (int i = 0; i < segmentCount; i++)
00509             {
00510                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
00511             }
00512         }
00513 
00514         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00515         {
00516             double length = this.Measure(previousPoint.Value);
00517             int segmentCount = (int)Math.Ceiling(length / resolution);
00518 
00519             for (int i = 0; i < segmentCount; i++)
00520             {
00521                 yield return this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount);
00522             }
00523         }
00524 
00525         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00526         {
00527             yield return new CubicBezierSegment(transformationFunction(this.Points[0]),
      transformationFunction(this.Points[1]), transformationFunction(this.Points[2]));
00528         }
00529 
00530         private (CubicBezierSegment first, CubicBezierSegment second) Subdivide(Point previousPoint,
      double t)
00531         {
00532             Point p0 = previousPoint + t * (this.Points[0] - previousPoint);
00533             Point p1 = this.Points[0] + t * (this.Points[1] - this.Points[0]);
00534             Point p2 = this.Points[1] + t * (this.Points[2] - this.Points[1]);
00535             Point p0_2 = p0 + t * (p1 - p0);
00536             Point p1_2 = p1 + t * (p2 - p1);
00537             Point p0_3 = p0_2 + t * (p1_2 - p0_2);
00538 
00539             return (new CubicBezierSegment(p0, p0_2, p0_3), new CubicBezierSegment(p1_2, p2,
      this.Points[2]));
00540         }
00541 
00542         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00543         private IEnumerable<Segment> FlattenPrivate(Point?  previousPoint, double flatness)
00544         {
00545             Point startingPoint;
```

```
00546
00547             startingPoint = previousPoint ??  this.Points[0];
00548
00549             double s2 = ((this.Points[1].X - startingPoint.X) * (this.Points[0].Y - startingPoint.Y) -
      (this.Points[1].Y - startingPoint.Y) * (this.Points[0].X - startingPoint.X)) /
      Math.Sqrt((this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X) +
      (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y));
00550
00551             double t = 2 * Math.Sqrt(flatness / (3 * Math.Abs(s2)));
00552
00553             if (t < 1)
00554             {
00555                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      t);
00556
00557                 yield return new LineSegment(first.Points[2]);
00558
00559                 foreach (Segment seg in second.FlattenPrivate(first.Points[2], flatness))
00560                 {
00561                     yield return seg;
00562                 }
00563             }
00564             else
00565             {
00566                 yield return new LineSegment(this.Points[2]);
00567             }
00568         }
00569
00570         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00571         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00572         {
00573             return FlattenForOffsetAndGetTangents(previousPoint, offset, flatness, false);
00574         }
00575
00576         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00577         private IEnumerable<(Point point, Point tangent)> FlattenForOffsetAndGetTangents(Point?
      previousPoint, double offset, double flatness, bool skipFirst)
00578         {
00579             Point startingPoint;
00580
00581             startingPoint = previousPoint ??  this.Points[0];
00582
00583             if (!skipFirst)
00584             {
00585                 yield return (startingPoint, this.GetBezierTangentAt(startingPoint, 0));
00586             }
00587
00588             double r1sq = (this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X)
      + (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y);
00589             double s2 = ((this.Points[1].X - startingPoint.X) * (this.Points[0].Y - startingPoint.Y) -
      (this.Points[1].Y - startingPoint.Y) * (this.Points[0].X - startingPoint.X)) /
      Math.Sqrt((this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X) +
      (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y));
00590
00591             if (double.IsNaN(s2))
00592             {
00593                 s2 = ((this.Points[2].X - startingPoint.X) * (this.Points[1].Y - startingPoint.Y) -
      (this.Points[2].Y - startingPoint.Y) * (this.Points[1].X - startingPoint.X)) /
      Math.Sqrt((this.Points[1].X - startingPoint.X) * (this.Points[1].X - startingPoint.X) +
      (this.Points[1].Y - startingPoint.Y) * (this.Points[1].Y - startingPoint.Y));
00594             }
00595
00596             if (double.IsNaN(s2))
00597             {
00598                 s2 = 1;
00599             }
00600
00601             double t = 2 * Math.Sqrt(Math.Abs(flatness / (3 * Math.Abs(s2) * (1 - offset * s2 / (3 *
      r1sq)))));
00602
00603             if (r1sq < 1e-7)
00604             {
00605                 t = 2 * Math.Sqrt(Math.Abs(flatness / (3 * Math.Abs(s2))));
00606
00607                 if (this.Points[1].X == startingPoint.X && this.Points[1].Y == startingPoint.Y &&
      this.Points[2].X == startingPoint.X && this.Points[2].Y == startingPoint.Y)
00608                 {
00609                     yield return (this.Point, new Point(double.NaN, double.NaN));
00610                     yield break;
00611                 }
00612             }
00613
00614             if (t < 1)
00615             {
00616                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      t);
```

```
00617
00618                 yield return (first.Points[2], this.GetBezierTangentAt(startingPoint, t));
00619
00620                 foreach ((Point, Point) p in second.FlattenForOffsetAndGetTangents(first.Points[2],
      offset, flatness, true))
00621                 {
00622                     yield return p;
00623                 }
00624             }
00625             else
00626             {
00627                 yield return (this.Points[2], this.GetBezierTangentAt(startingPoint, 1));
00628             }
00629         }
00630
00631         internal IEnumerable<CubicBezierSegment> MonotoniseOnY(Point previousPoint)
00632         {
00633             double a = previousPoint.Y;
00634             double b = this.Points[0].Y;
00635             double c = this.Points[1].Y;
00636             double d = this.Points[2].Y;
00637
00638             if ((a + 3 * c != 3 * b + d))
00639             {
00640                 double t1 = ((-6 * a + 12 * b - 6 * c) + Math.Sqrt((6 * a - 12 * b + 6 * c) * (6 * a -
      12 * b + 6 * c) - 4 * (3 * b - 3 * a) * (-3 * a + 9 * b - 9 * c + 3 * d))) / (2 * (-3 * a + 9 * b - 9
      * c + 3 * d));
00641                 double t2 = ((-6 * a + 12 * b - 6 * c) - Math.Sqrt((6 * a - 12 * b + 6 * c) * (6 * a -
      12 * b + 6 * c) - 4 * (3 * b - 3 * a) * (-3 * a + 9 * b - 9 * c + 3 * d))) / (2 * (-3 * a + 9 * b - 9
      * c + 3 * d));
00642
00643                 if (t1 > 0 && t1 < 1 && t2 > 0 && t2 < 1)
00644                 {
00645                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      Math.Min(t1, t2));
00646
00647                     yield return seg1;
00648
00649                     foreach (CubicBezierSegment seg in seg2.MonotoniseOnY(seg1.Point))
00650                     {
00651                         yield return seg;
00652                     }
00653                 }
00654                 else if (t1 > 0 && t1 < 1)
00655                 {
00656                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t1);
00657
00658                     yield return seg1;
00659                     yield return seg2;
00660                 }
00661                 else if (t2 > 0 && t2 < 1)
00662                 {
00663                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t2);
00664
00665                     yield return seg1;
00666                     yield return seg2;
00667                 }
00668                 else
00669                 {
00670                     yield return this;
00671                 }
00672             }
00673             else
00674             {
00675                 double t = (a - b) / (2 * (a - 2 * b + c));
00676
00677                 if (t > 0 && t < 1)
00678                 {
00679                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t);
00680
00681                     yield return seg1;
00682                     yield return seg2;
00683                 }
00684                 else
00685                 {
00686                     yield return this;
00687                 }
00688             }
00689         }
00690
00691         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00692         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00693         {
00694             Point currPoint = previousPoint ??  this.Points[0];
```

```
00695
00696             foreach (Segment seg in this.BreakAtInflectionPoints(previousPoint, flatness))
00697             {
00698                 if (seg is CubicBezierSegment cub)
00699                 {
00700                     foreach (Segment seg2 in cub.FlattenPrivate(currPoint, flatness))
00701                     {
00702                         yield return seg2;
00703                     }
00704                 }
00705                 else
00706                 {
00707                     yield return seg;
00708                 }
00709                 currPoint = seg.Point;
00710             }
00711         }
00712
00713         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00714         internal IEnumerable<Segment> BreakAtInflectionPoints(Point?  previousPoint, double flatness)
00715         {
00716             Point startingPoint;
00717
00718             startingPoint = previousPoint ??  this.Points[0];
00719
00720             double ax = -startingPoint.X + 3 * this.Points[0].X - 3 * this.Points[1].X +
      this.Points[2].X;
00721             double ay = -startingPoint.Y + 3 * this.Points[0].Y - 3 * this.Points[1].Y +
      this.Points[2].Y;
00722             double bx = 3 * startingPoint.X - 6 * this.Points[0].X + 3 * this.Points[1].X;
00723             double by = 3 * startingPoint.Y - 6 * this.Points[0].Y + 3 * this.Points[1].Y;
00724             double cx = -3 * startingPoint.X + 3 * this.Points[0].X;
00725             double cy = -3 * startingPoint.Y + 3 * this.Points[0].Y;
00726
00727             double tcusp = -0.5 * (ay * cx - ax * cy) / (ay * bx - ax * by);
00728
00729             double t1 = tcusp - Math.Sqrt(tcusp * tcusp - (by * cx - bx * cy) / (ay * bx - ax * by) /
      3);
00730             double t2 = tcusp + Math.Sqrt(tcusp * tcusp - (by * cx - bx * cy) / (ay * bx - ax * by) /
      3);
00731
00732             List<double> validInflectionPoints = new List<double>();
00733
00734             if (t1 >= 0 && t1 <= 1)
00735             {
00736                 validInflectionPoints.Add(t1);
00737             }
00738
00739             if (t2 >= 0 && t2 <= 1 && (validInflectionPoints.Count == 0 || t2 -
      validInflectionPoints[0] > 1e-5))
00740             {
00741                 validInflectionPoints.Add(t2);
00742             }
00743
00744             if (validInflectionPoints.Count == 0)
00745             {
00746                 yield return this;
00747             }
00748             else if (validInflectionPoints.Count == 1)
00749             {
00750                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      validInflectionPoints[0]);
00751
00752                 double s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00753
00754                 double tf = Math.Pow(flatness / s3, 1.0 / 3);
00755
00756                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00757                 {
00758                     tf = 0;
00759                 }
00760
00761                 double tm = Math.Max(0, validInflectionPoints[0] - tf * (1 -
      validInflectionPoints[0]));
00762                 double tp = Math.Min(validInflectionPoints[0] + tf * (1 - validInflectionPoints[0]),
      1);
00763
00764                 if (tm > 0 && tp < 1)
00765                 {
00766                     (first, _) = this.Subdivide(startingPoint, tm);
00767
00768                     CubicBezierSegment temp;
00769                     (temp, second) = this.Subdivide(startingPoint, tp);
00770
```

```
00771                     yield return first;
00772
00773                     if (tm != tp)
00774                     {
00775                         yield return new LineSegment(temp.Point);
00776                     }
00777
00778                     yield return second;
00779                 }
00780                 else if (tm == 0 && tp < 1)
00781                 {
00782                     (first, second) = this.Subdivide(startingPoint, tp);
00783
00784                     yield return new LineSegment(first.Point);
00785
00786                     yield return second;
00787                 }
00788                 else if (tm > 0 && tp == 1)
00789                 {
00790                     (first, _) = this.Subdivide(startingPoint, tm);
00791
00792                     yield return first;
00793
00794                     if (tm != tp)
00795                     {
00796                         yield return new LineSegment(this.Point);
00797                     }
00798                 }
00799                 else
00800                 {
00801                     yield return new LineSegment(this.Point);
00802                 }
00803             }
00804             else if (validInflectionPoints.Count == 2)
00805             {
00806                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      validInflectionPoints[0]);
00807
00808                 double s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00809
00810                 double tf = Math.Pow(flatness / s3, 1.0 / 3);
00811
00812                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00813                 {
00814                     tf = 0;
00815                 }
00816
00817                 double t1m = Math.Max(validInflectionPoints[0] - tf * (1 - validInflectionPoints[0]),
      0);
00818                 double t1p = Math.Min(validInflectionPoints[0] + tf * (1 - validInflectionPoints[0]),
      validInflectionPoints[1]);
00819
00820
00821                 (first, second) = this.Subdivide(startingPoint, validInflectionPoints[1]);
00822
00823                 s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00824
00825                 tf = Math.Pow(flatness / s3, 1.0 / 3);
00826
00827                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00828                 {
00829                     tf = 0;
00830                 }
00831
00832                 double t2m = Math.Max(validInflectionPoints[1] - tf * (1 - validInflectionPoints[1]),
      t1p);
00833                 double t2p = Math.Min(validInflectionPoints[1] + tf * (1 - validInflectionPoints[1]),
      1);
00834
00835                 if (t1m > 0)
00836                 {
00837                     (first, _) = this.Subdivide(startingPoint, t1m);
00838                     yield return first;
00839                 }
00840
00841                 if (t1p > t1m)
00842                 {
00843                     CubicBezierSegment temp;
00844                     (temp, _) = this.Subdivide(startingPoint, t1p);
00845                     yield return new LineSegment(temp.Point);
00846                 }
```

```
00847
00848                 if (t2m > t1p)
00849                 {
00850                     CubicBezierSegment temp, third;
00851                     (temp, second) = this.Subdivide(startingPoint, t1p);
00852                     (third, _) = second.Subdivide(temp.Point, (t2m - t1p) / (1 - t1p));
00853                     yield return third;
00854                 }
00855
00856                 if (t2p > t2m)
00857                 {
00858                     CubicBezierSegment temp;
00859                     (temp, second) = this.Subdivide(startingPoint, t1p);
00860                     (temp, _) = second.Subdivide(temp.Point, (t2p - t1p) / (1 - t1p));
00861                     yield return new LineSegment(temp.Point);
00862                 }
00863
00864                 if (t2p < 1)
00865                 {
00866                     CubicBezierSegment temp, fourth;
00867                     (temp, second) = this.Subdivide(startingPoint, t1p);
00868                     (_, fourth) = second.Subdivide(temp.Point, (t2p - t1p) / (1 - t1p));
00869                     yield return fourth;
00870                 }
00871             }
00872         }
00873     }
00874
00875     internal class ArcSegment :  Segment
00876     {
00877         public override SegmentType Type => SegmentType.Arc;
00878
00879         public Segment[] ToBezierSegments()
00880         {
00881             List<Segment> tbr = new List<Segment>();
00882
00883             if (EndAngle > StartAngle)
00884             {
00885                 if (EndAngle - StartAngle <= Math.PI / 2)
00886                 {
00887                     tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, StartAngle,
      EndAngle, true));
00888                 }
00889                 else
00890                 {
00891                     int count = (int)Math.Ceiling(2 * (EndAngle - StartAngle) / Math.PI);
00892                     double angle = StartAngle;
00893
00894                     for (int i = 0; i < count; i++)
00895                     {
00896                         tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, angle, angle +
      (EndAngle - StartAngle) / count, i == 0));
00897                         angle += (EndAngle - StartAngle) / count;
00898                     }
00899                 }
00900             }
00901             else if (EndAngle < StartAngle)
00902             {
00903                 Point startPoint = new Point(Points[0].X + Radius * Math.Cos(EndAngle), Points[0].Y +
      Radius * Math.Sin(EndAngle));
00904                 if (StartAngle - EndAngle <= Math.PI / 2)
00905                 {
00906                     tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, EndAngle,
      StartAngle, true));
00907                 }
00908                 else
00909                 {
00910                     int count = (int)Math.Ceiling(2 * (StartAngle - EndAngle) / Math.PI);
00911                     double angle = EndAngle;
00912
00913                     for (int i = 0; i < count; i++)
00914                     {
00915                         tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, angle, angle +
      (StartAngle - EndAngle) / count, i == 0));
00916                         angle += (StartAngle - EndAngle) / count;
00917                     }
00918                 }
00919
00920                 return ReverseSegments(tbr, startPoint).ToArray();
00921             }
00922
00923             return tbr.ToArray();
00924         }
00925
00926         private static Segment[] ReverseSegments(IReadOnlyList<Segment> originalSegments, Point
      startPoint)
00927         {
```

```
00928             List<Segment> tbr = new List<Segment>(originalSegments.Count);
00929
00930             for (int i = originalSegments.Count - 1; i >= 0; i--)
00931             {
00932                 switch (originalSegments[i].Type)
00933                 {
00934                     case SegmentType.Line:
00935                         if (i > 0)
00936                         {
00937                             tbr.Add(new LineSegment(originalSegments[i - 1].Point));
00938                         }
00939                         else
00940                         {
00941                             tbr.Add(new LineSegment(startPoint));
00942                         }
00943                         break;
00944                     case SegmentType.CubicBezier:
00945                         CubicBezierSegment originalSegment = (CubicBezierSegment)originalSegments[i];
00946                         if (i > 0)
00947                         {
00948                             tbr.Add(new CubicBezierSegment(originalSegment.Points[1],
      originalSegment.Points[0], originalSegments[i - 1].Point));
00949                         }
00950                         else
00951                         {
00952                             tbr.Add(new CubicBezierSegment(originalSegment.Points[1],
      originalSegment.Points[0], startPoint));
00953                         }
00954                         break;
00955                 }
00956             }
00957
00958             return tbr.ToArray();
00959         }
00960
00961         const double k = 0.55191496;
00962
00963         private static Segment[] GetBezierSegment(double cX, double cY, double radius, double
      startAngle, double endAngle, bool firstArc)
00964         {
00965             double phi = Math.PI / 4;
00966
00967             double x1 = radius * Math.Cos(phi);
00968             double y1 = radius * Math.Sin(phi);
00969
00970             double x4 = x1;
00971             double y4 = -y1;
00972
00973             double x3 = x1 + k * radius * Math.Sin(phi);
00974             double y3 = y1 - k * radius * Math.Cos(phi);
00975
00976             double x2 = x4 + k * radius * Math.Sin(phi);
00977             double y2 = y4 + k * radius * Math.Cos(phi);
00978
00979             double u = 2 * (endAngle - startAngle) / Math.PI;
00980
00981             double fx2 = (1 - u) * x4 + u * x2;
00982             double fy2 = (1 - u) * y4 + u * y2;
00983
00984             double fx3 = (1 - u) * fx2 + u * ((1 - u) * x2 + u * x3);
00985             double fy3 = (1 - u) * fy2 + u * ((1 - u) * y2 + u * y3);
00986
00987             double rX1 = cX + radius * Math.Cos(startAngle);
00988             double rY1 = cY + radius * Math.Sin(startAngle);
00989
00990             double rX4 = cX + radius * Math.Cos(endAngle);
00991             double rY4 = cY + radius * Math.Sin(endAngle);
00992
00993             Point rot2 = Utils.RotatePoint(new Point(fx2, fy2), phi + startAngle);
00994             Point rot3 = Utils.RotatePoint(new Point(fx3, fy3), phi + startAngle);
00995
00996             List<Segment> tbr = new List<Segment>();
00997
00998             if (firstArc)
00999             {
01000                 tbr.Add(new LineSegment(rX1, rY1));
01001             }
01002
01003             tbr.Add(new CubicBezierSegment(cX + rot2.X, cY + rot2.Y, cX + rot3.X, cY + rot3.Y, rX4,
      rY4));
01004
01005             return tbr.ToArray();
01006         }
01007         public double Radius { get; }
01008         public double StartAngle { get; }
01009         public double EndAngle { get; }
01010
```

```
01011         public ArcSegment(Point center, double radius, double startAngle, double endAngle)
01012         {
01013             this.Points = new Point[] { center };
01014             this.Radius = radius;
01015             this.StartAngle = startAngle;
01016             this.EndAngle = endAngle;
01017         }
01018 
01019         public ArcSegment(double centerX, double centerY, double radius, double startAngle, double
      endAngle)
01020         {
01021             this.Points = new Point[] { new Point(centerX, centerY) };
01022             this.Radius = radius;
01023             this.StartAngle = startAngle;
01024             this.EndAngle = endAngle;
01025         }
01026 
01027         public override Segment Clone()
01028         {
01029             return new ArcSegment(Point.X, Point.Y, Radius, StartAngle, EndAngle);
01030         }
01031 
01032         public override Point Point
01033         {
01034             get
01035             {
01036                 return new Point(this.Points[0].X + Math.Cos(EndAngle) * Radius, this.Points[0].Y +
      Math.Sin(EndAngle) * Radius);
01037             }
01038         }
01039 
01040 
01041         private double cachedLength = double.NaN;
01042 
01043         public override double Measure(Point previousPoint)
01044         {
01045             if (double.IsNaN(cachedLength))
01046             {
01047                 Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01048 
01049                 cachedLength = Radius * Math.Abs(EndAngle - StartAngle) + Math.Sqrt((arcStartPoint.X -
      previousPoint.X) * (arcStartPoint.X - previousPoint.X) + (arcStartPoint.Y - previousPoint.Y) *
      (arcStartPoint.Y - previousPoint.Y));
01050             }
01051 
01052             return cachedLength;
01053         }
01054 
01055         public override Point GetPointAt(Point previousPoint, double position)
01056         {
01057             double totalLength = this.Measure(previousPoint);
01058             double arcLength = Radius * Math.Abs(EndAngle - StartAngle);
01059 
01060             double preArc = (totalLength - arcLength) / totalLength;
01061 
01062             if (position < preArc)
01063             {
01064                 if (position >= 0)
01065                 {
01066                     double relPos = position / preArc;
01067                     Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01068 
01069                     return new Point(previousPoint.X * (1 - relPos) + arcStartPoint.X * relPos,
      previousPoint.Y * (1 - relPos) + arcStartPoint.Y * relPos);
01070                 }
01071                 else
01072                 {
01073                     Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01074                     Point tangent = GetTangentAt(previousPoint, 0);
01075                     double excessLength = position * this.Measure(previousPoint);
01076                     return new Point(arcStartPoint.X + tangent.X * excessLength, arcStartPoint.Y +
      tangent.Y * excessLength);
01077                 }
01078             }
01079             else
01080             {
01081                 double relPos = position - preArc / (1 - preArc);
01082 
01083                 if (relPos <= 1)
01084                 {
01085                     double angle = StartAngle * (1 - relPos) + EndAngle * relPos;
01086                     return new Point(this.Points[0].X + Radius * Math.Cos(angle), this.Points[0].Y +
      Radius * Math.Sin(angle));
01087                 }
```

```
01088                 else
01089                 {
01090                     Point arcEndPoint = this.Point;
01091                     Point tangent = GetTangentAt(previousPoint, 1);
01092                     double excessLength = (position - 1) * this.Measure(previousPoint);
01093                     return new Point(arcEndPoint.X + tangent.X * excessLength, arcEndPoint.Y +
      tangent.Y * excessLength);
01094                 }
01095             }
01096         }
01097 
01098         public override Point GetTangentAt(Point previousPoint, double position)
01099         {
01100             double totalLength = this.Measure(previousPoint);
01101             double arcLength = Radius * Math.Abs(EndAngle - StartAngle);
01102 
01103             double preArc = (totalLength - arcLength) / totalLength;
01104 
01105             if (position < preArc)
01106             {
01107                 Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01108                 Point tang = new Point((arcStartPoint.X - previousPoint.X) * Math.Sign(EndAngle -
      StartAngle), (arcStartPoint.Y - previousPoint.Y) * Math.Sign(EndAngle - StartAngle)).Normalize();
01109 
01110                 if (tang.Modulus() > 0.001)
01111                 {
01112                     return tang.Normalize();
01113                 }
01114                 else
01115                 {
01116                     return this.GetTangentAt(previousPoint, 0);
01117                 }
01118             }
01119             else
01120             {
01121                 double relPos = position - preArc / (1 - preArc);
01122 
01123                 if (relPos <= 1)
01124                 {
01125                     double angle = StartAngle * (1 - relPos) + EndAngle * relPos;
01126                     return new Point(-Math.Sin(angle) * Math.Sign(EndAngle - StartAngle),
      Math.Cos(angle) * Math.Sign(EndAngle - StartAngle));
01127                 }
01128                 else
01129                 {
01130                     return new Point(-Math.Sin(EndAngle) * Math.Sign(EndAngle - StartAngle),
      Math.Cos(EndAngle) * Math.Sign(EndAngle - StartAngle));
01131                 }
01132             }
01133 
01134         }
01135 
01136         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
01137         {
01138             double length = this.Measure(previousPoint.Value);
01139             int segmentCount = (int)Math.Ceiling(length / resolution);
01140 
01141             for (int i = 0; i < segmentCount; i++)
01142             {
01143                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
01144             }
01145         }
01146         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
01147         {
01148             double length = this.Measure(previousPoint.Value);
01149             int segmentCount = (int)Math.Ceiling(length / resolution);
01150 
01151             for (int i = 0; i < segmentCount; i++)
01152             {
01153                 yield return this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount);
01154             }
01155         }
01156 
01157         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
01158         {
01159             foreach (Segment seg in this.ToBezierSegments())
01160             {
01161                 foreach (Segment seg2 in seg.Transform(transformationFunction))
01162                 {
01163                     yield return seg2;
01164                 }
01165             }
01166         }
01167 
```

```
01168         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
01169         {
01170             double length = this.Measure(previousPoint.Value);
01171             double resolution = 2 * Math.Sqrt(flatness * (2 * this.Radius - flatness));
01172 
01173             int segmentCount = (int)Math.Ceiling(length / resolution);
01174 
01175             for (int i = 0; i < segmentCount; i++)
01176             {
01177                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
01178             }
01179         }
01180 
01181         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
01182         {
01183             double length = this.Measure(previousPoint.Value);
01184             double resolution = 2 * Math.Sqrt(flatness * (2 * (this.Radius + offset) - flatness));
01185 
01186             int segmentCount = (int)Math.Ceiling(length / resolution);
01187 
01188             yield return (this.GetPointAt(previousPoint.Value, 0),
      this.GetTangentAt(previousPoint.Value, 0));
01189 
01190             for (int i = 0; i < segmentCount; i++)
01191             {
01192                 yield return (this.GetPointAt(previousPoint.Value, (double)(i + 1) / segmentCount),
      this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount));
01193             }
01194         }
01195     }
01196 
01197 
01198 }
```

8.74 SmoothSpline.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System.Collections.Generic;
00019 using System.Linq;
00020 
00021 namespace VectSharp
00022 {
00023     //derived from https://www.particleincell.com/wp-content/uploads/2012/06/bezier-spline.js
00024 
00025     internal static class SmoothSpline
00026     {
00027         public static Point[] SmoothSplines(Point[] points)
00028         {
00029             double[] x = (from el in points select el.X).ToArray();
00030             double[] y = (from el in points select el.Y).ToArray();
00031 
00032             (double[] p1, double[] p2) px = ComputeControlPoints(x);
00033             (double[] p1, double[] p2) py = ComputeControlPoints(y);
00034 
00035             List<Point> tbr = new List<Point>();
00036 
00037             for (int i = 0; i < points.Length - 1; i++)
00038             {
00039                 tbr.Add(new Point(x[i], y[i]));
00040                 tbr.Add(new Point(px.p1[i], py.p1[i]));
00041                 tbr.Add(new Point(px.p2[i], py.p2[i]));
00042             }
00043 
00044             tbr.Add(new Point(x[x.Length - 1], y[x.Length - 1]));
00045 
```

```
00046             return tbr.ToArray();
00047         }
00048 
00049         /*computes control points given knots K, this is the brain of the operation*/
00050         private static (double[] p1, double[] p2) ComputeControlPoints(double[] K)
00051         {
00052             int n = K.Length - 1;
00053 
00054 
00055             double[] p1 = new double[n];
00056             double[] p2 = new double[n];
00057 
00058 
00059             /*rhs vector*/
00060             double[] a = new double[n];
00061             double[] b = new double[n];
00062             double[] c = new double[n];
00063             double[] r = new double[n];
00064 
00065             /*left most segment*/
00066             a[0] = 0;
00067             b[0] = 2;
00068             c[0] = 1;
00069             r[0] = K[0] + 2 * K[1];
00070 
00071             /*internal segments*/
00072             for (int i = 1; i < n - 1; i++)
00073             {
00074                 a[i] = 1;
00075                 b[i] = 4;
00076                 c[i] = 1;
00077                 r[i] = 4 * K[i] + 2 * K[i + 1];
00078             }
00079 
00080             /*right segment*/
00081             a[n - 1] = 2;
00082             b[n - 1] = 7;
00083             c[n - 1] = 0;
00084             r[n - 1] = 8 * K[n - 1] + K[n];
00085 
00086             /*solves Ax=b with the Thomas algorithm (from Wikipedia)*/
00087             for (int i = 1; i < n; i++)
00088             {
00089                 double m = a[i] / b[i - 1];
00090                 b[i] = b[i] - m * c[i - 1];
00091                 r[i] = r[i] - m * r[i - 1];
00092             }
00093 
00094             p1[n - 1] = r[n - 1] / b[n - 1];
00095             for (int i = n - 2; i >= 0; i--)
00096             {
00097                 p1[i] = (r[i] - c[i] * p1[i + 1]) / b[i];
00098             }
00099 
00100             /*we have p1, now compute p2*/
00101             for (int i = 0; i < n - 1; i++)
00102             {
00103                 p2[i] = 2 * K[i + 1] - p1[i + 1];
00104             }
00105 
00106             p2[n - 1] = 0.5 * (K[n] + p1[n - 1]);
00107 
00108             return (p1, p2);
00109         }
00110 
00111     }
00112 }
```

8.75 StandardColours.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
```

```
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021
00022 namespace VectSharp
00023 {
00024     public partial struct Colour
00025     {
00026         private static Dictionary<string, Colour> StandardColours = new Dictionary<string,
      Colour>(StringComparer.OrdinalIgnoreCase)
00027         {
00028             { "Black", Colour.FromRgb(0, 0, 0) },
00029             { "Navy", Colour.FromRgb(0, 0, 128) },
00030             { "DarkBlue", Colour.FromRgb(0, 0, 139) },
00031             { "MediumBlue", Colour.FromRgb(0, 0, 205) },
00032             { "Blue", Colour.FromRgb(0, 0, 255) },
00033             { "DarkGreen", Colour.FromRgb(0, 100, 0) },
00034             { "Green", Colour.FromRgb(0, 128, 0) },
00035             { "Teal", Colour.FromRgb(0, 128, 128) },
00036             { "DarkCyan", Colour.FromRgb(0, 139, 139) },
00037             { "DeepSkyBlue", Colour.FromRgb(0, 191, 255) },
00038             { "DarkTurquoise", Colour.FromRgb(0, 206, 209) },
00039             { "MediumSpringGreen", Colour.FromRgb(0, 250, 154) },
00040             { "Lime", Colour.FromRgb(0, 255, 0) },
00041             { "SpringGreen", Colour.FromRgb(0, 255, 127) },
00042             { "Aqua", Colour.FromRgb(0, 255, 255) },
00043             { "Cyan", Colour.FromRgb(0, 255, 255) },
00044             { "MidnightBlue", Colour.FromRgb(25, 25, 112) },
00045             { "DodgerBlue", Colour.FromRgb(30, 144, 255) },
00046             { "LightSeaGreen", Colour.FromRgb(32, 178, 170) },
00047             { "ForestGreen", Colour.FromRgb(34, 139, 34) },
00048             { "SeaGreen", Colour.FromRgb(46, 139, 87) },
00049             { "DarkSlateGray", Colour.FromRgb(47, 79, 79) },
00050             { "DarkSlateGrey", Colour.FromRgb(47, 79, 79) },
00051             { "LimeGreen", Colour.FromRgb(50, 205, 50) },
00052             { "MediumSeaGreen", Colour.FromRgb(60, 179, 113) },
00053             { "Turquoise", Colour.FromRgb(64, 224, 208) },
00054             { "RoyalBlue", Colour.FromRgb(65, 105, 225) },
00055             { "SteelBlue", Colour.FromRgb(70, 130, 180) },
00056             { "DarkSlateBlue", Colour.FromRgb(72, 61, 139) },
00057             { "MediumTurquoise", Colour.FromRgb(72, 209, 204) },
00058             { "Indigo", Colour.FromRgb(75, 0, 130) },
00059             { "DarkOliveGreen", Colour.FromRgb(85, 107, 47) },
00060             { "CadetBlue", Colour.FromRgb(95, 158, 160) },
00061             { "CornflowerBlue", Colour.FromRgb(100, 149, 237) },
00062             { "RebeccaPurple", Colour.FromRgb(102, 51, 153) },
00063             { "MediumAquaMarine", Colour.FromRgb(102, 205, 170) },
00064             { "DimGray", Colour.FromRgb(105, 105, 105) },
00065             { "DimGrey", Colour.FromRgb(105, 105, 105) },
00066             { "SlateBlue", Colour.FromRgb(106, 90, 205) },
00067             { "OliveDrab", Colour.FromRgb(107, 142, 35) },
00068             { "SlateGray", Colour.FromRgb(112, 128, 144) },
00069             { "SlateGrey", Colour.FromRgb(112, 128, 144) },
00070             { "LightSlateGray", Colour.FromRgb(119, 136, 153) },
00071             { "LightSlateGrey", Colour.FromRgb(119, 136, 153) },
00072             { "MediumSlateBlue", Colour.FromRgb(123, 104, 238) },
00073             { "LawnGreen", Colour.FromRgb(124, 252, 0) },
00074             { "Chartreuse", Colour.FromRgb(127, 255, 0) },
00075             { "Aquamarine", Colour.FromRgb(127, 255, 212) },
00076             { "Maroon", Colour.FromRgb(128, 0, 0) },
00077             { "Purple", Colour.FromRgb(128, 0, 128) },
00078             { "Olive", Colour.FromRgb(128, 128, 0) },
00079             { "Gray", Colour.FromRgb(128, 128, 128) },
00080             { "Grey", Colour.FromRgb(128, 128, 128) },
00081             { "SkyBlue", Colour.FromRgb(135, 206, 235) },
00082             { "LightSkyBlue", Colour.FromRgb(135, 206, 250) },
00083             { "BlueViolet", Colour.FromRgb(138, 43, 226) },
00084             { "DarkRed", Colour.FromRgb(139, 0, 0) },
00085             { "DarkMagenta", Colour.FromRgb(139, 0, 139) },
00086             { "SaddleBrown", Colour.FromRgb(139, 69, 19) },
00087             { "DarkSeaGreen", Colour.FromRgb(143, 188, 143) },
00088             { "LightGreen", Colour.FromRgb(144, 238, 144) },
00089             { "MediumPurple", Colour.FromRgb(147, 112, 219) },
00090             { "DarkViolet", Colour.FromRgb(148, 0, 211) },
00091             { "PaleGreen", Colour.FromRgb(152, 251, 152) },
00092             { "DarkOrchid", Colour.FromRgb(153, 50, 204) },
00093             { "YellowGreen", Colour.FromRgb(154, 205, 50) },
00094             { "Sienna", Colour.FromRgb(160, 82, 45) },
00095             { "Brown", Colour.FromRgb(165, 42, 42) },
00096             { "DarkGray", Colour.FromRgb(169, 169, 169) },
00097             { "DarkGrey", Colour.FromRgb(169, 169, 169) },
00098             { "LightBlue", Colour.FromRgb(173, 216, 230) },
00099             { "GreenYellow", Colour.FromRgb(173, 255, 47) },
```

```
00100             { "PaleTurquoise", Colour.FromRgb(175, 238, 238) },
00101             { "LightSteelBlue", Colour.FromRgb(176, 196, 222) },
00102             { "PowderBlue", Colour.FromRgb(176, 224, 230) },
00103             { "FireBrick", Colour.FromRgb(178, 34, 34) },
00104             { "DarkGoldenRod", Colour.FromRgb(184, 134, 11) },
00105             { "MediumOrchid", Colour.FromRgb(186, 85, 211) },
00106             { "RosyBrown", Colour.FromRgb(188, 143, 143) },
00107             { "DarkKhaki", Colour.FromRgb(189, 183, 107) },
00108             { "Silver", Colour.FromRgb(192, 192, 192) },
00109             { "MediumVioletRed", Colour.FromRgb(199, 21, 133) },
00110             { "IndianRed", Colour.FromRgb(205, 92, 92) },
00111             { "Peru", Colour.FromRgb(205, 133, 63) },
00112             { "Chocolate", Colour.FromRgb(210, 105, 30) },
00113             { "Tan", Colour.FromRgb(210, 180, 140) },
00114             { "LightGray", Colour.FromRgb(211, 211, 211) },
00115             { "LightGrey", Colour.FromRgb(211, 211, 211) },
00116             { "Thistle", Colour.FromRgb(216, 191, 216) },
00117             { "Orchid", Colour.FromRgb(218, 112, 214) },
00118             { "GoldenRod", Colour.FromRgb(218, 165, 32) },
00119             { "PaleVioletRed", Colour.FromRgb(219, 112, 147) },
00120             { "Crimson", Colour.FromRgb(220, 20, 60) },
00121             { "Gainsboro", Colour.FromRgb(220, 220, 220) },
00122             { "Plum", Colour.FromRgb(221, 160, 221) },
00123             { "BurlyWood", Colour.FromRgb(222, 184, 135) },
00124             { "LightCyan", Colour.FromRgb(224, 255, 255) },
00125             { "Lavender", Colour.FromRgb(230, 230, 250) },
00126             { "DarkSalmon", Colour.FromRgb(233, 150, 122) },
00127             { "Violet", Colour.FromRgb(238, 130, 238) },
00128             { "PaleGoldenRod", Colour.FromRgb(238, 232, 170) },
00129             { "LightCoral", Colour.FromRgb(240, 128, 128) },
00130             { "Khaki", Colour.FromRgb(240, 230, 140) },
00131             { "AliceBlue", Colour.FromRgb(240, 248, 255) },
00132             { "HoneyDew", Colour.FromRgb(240, 255, 240) },
00133             { "Azure", Colour.FromRgb(240, 255, 255) },
00134             { "SandyBrown", Colour.FromRgb(244, 164, 96) },
00135             { "Wheat", Colour.FromRgb(245, 222, 179) },
00136             { "Beige", Colour.FromRgb(245, 245, 220) },
00137             { "WhiteSmoke", Colour.FromRgb(245, 245, 245) },
00138             { "MintCream", Colour.FromRgb(245, 255, 250) },
00139             { "GhostWhite", Colour.FromRgb(248, 248, 255) },
00140             { "Salmon", Colour.FromRgb(250, 128, 114) },
00141             { "AntiqueWhite", Colour.FromRgb(250, 235, 215) },
00142             { "Linen", Colour.FromRgb(250, 240, 230) },
00143             { "LightGoldenRodYellow", Colour.FromRgb(250, 250, 210) },
00144             { "OldLace", Colour.FromRgb(253, 245, 230) },
00145             { "Red", Colour.FromRgb(255, 0, 0) },
00146             { "Fuchsia", Colour.FromRgb(255, 0, 255) },
00147             { "Magenta", Colour.FromRgb(255, 0, 255) },
00148             { "DeepPink", Colour.FromRgb(255, 20, 147) },
00149             { "OrangeRed", Colour.FromRgb(255, 69, 0) },
00150             { "Tomato", Colour.FromRgb(255, 99, 71) },
00151             { "HotPink", Colour.FromRgb(255, 105, 180) },
00152             { "Coral", Colour.FromRgb(255, 127, 80) },
00153             { "DarkOrange", Colour.FromRgb(255, 140, 0) },
00154             { "LightSalmon", Colour.FromRgb(255, 160, 122) },
00155             { "Orange", Colour.FromRgb(255, 165, 0) },
00156             { "LightPink", Colour.FromRgb(255, 182, 193) },
00157             { "Pink", Colour.FromRgb(255, 192, 203) },
00158             { "Gold", Colour.FromRgb(255, 215, 0) },
00159             { "PeachPuff", Colour.FromRgb(255, 218, 185) },
00160             { "NavajoWhite", Colour.FromRgb(255, 222, 173) },
00161             { "Moccasin", Colour.FromRgb(255, 228, 181) },
00162             { "Bisque", Colour.FromRgb(255, 228, 196) },
00163             { "MistyRose", Colour.FromRgb(255, 228, 225) },
00164             { "BlanchedAlmond", Colour.FromRgb(255, 235, 205) },
00165             { "PapayaWhip", Colour.FromRgb(255, 239, 213) },
00166             { "LavenderBlush", Colour.FromRgb(255, 240, 245) },
00167             { "SeaShell", Colour.FromRgb(255, 245, 238) },
00168             { "Cornsilk", Colour.FromRgb(255, 248, 220) },
00169             { "LemonChiffon", Colour.FromRgb(255, 250, 205) },
00170             { "FloralWhite", Colour.FromRgb(255, 250, 240) },
00171             { "Snow", Colour.FromRgb(255, 250, 250) },
00172             { "Yellow", Colour.FromRgb(255, 255, 0) },
00173             { "LightYellow", Colour.FromRgb(255, 255, 224) },
00174             { "Ivory", Colour.FromRgb(255, 255, 240) },
00175             { "White", Colour.FromRgb(255, 255, 255) },
00176         };
00177     }
00178
00179 /// <summary>
00180 /// Standard colours.
00181 /// </summary>
00182     public static class Colours
00183     {
00184 /// <summary>
00185 /// Black #000000
00186 /// </summary>
```

```
00187         public static Colour Black = Colour.FromRgb(0, 0, 0);
00188 /// <summary>
00189 /// Navy #000080
00190 /// </summary>
00191         public static Colour Navy = Colour.FromRgb(0, 0, 128);
00192 /// <summary>
00193 /// DarkBlue #00008B
00194 /// </summary>
00195         public static Colour DarkBlue = Colour.FromRgb(0, 0, 139);
00196 /// <summary>
00197 /// MediumBlue #0000CD
00198 /// </summary>
00199         public static Colour MediumBlue = Colour.FromRgb(0, 0, 205);
00200 /// <summary>
00201 /// Blue #0000FF
00202 /// </summary>
00203         public static Colour Blue = Colour.FromRgb(0, 0, 255);
00204 /// <summary>
00205 /// DarkGreen #006400
00206 /// </summary>
00207         public static Colour DarkGreen = Colour.FromRgb(0, 100, 0);
00208 /// <summary>
00209 /// Green #008000
00210 /// </summary>
00211         public static Colour Green = Colour.FromRgb(0, 128, 0);
00212 /// <summary>
00213 /// Teal #008080
00214 /// </summary>
00215         public static Colour Teal = Colour.FromRgb(0, 128, 128);
00216 /// <summary>
00217 /// DarkCyan #008B8B
00218 /// </summary>
00219         public static Colour DarkCyan = Colour.FromRgb(0, 139, 139);
00220 /// <summary>
00221 /// DeepSkyBlue #00BFFF
00222 /// </summary>
00223         public static Colour DeepSkyBlue = Colour.FromRgb(0, 191, 255);
00224 /// <summary>
00225 /// DarkTurquoise #00CED1
00226 /// </summary>
00227         public static Colour DarkTurquoise = Colour.FromRgb(0, 206, 209);
00228 /// <summary>
00229 /// MediumSpringGreen #00FA9A
00230 /// </summary>
00231         public static Colour MediumSpringGreen = Colour.FromRgb(0, 250, 154);
00232 /// <summary>
00233 /// Lime #00FF00
00234 /// </summary>
00235         public static Colour Lime = Colour.FromRgb(0, 255, 0);
00236 /// <summary>
00237 /// SpringGreen #00FF7F
00238 /// </summary>
00239         public static Colour SpringGreen = Colour.FromRgb(0, 255, 127);
00240 /// <summary>
00241 /// Aqua #00FFFF
00242 /// </summary>
00243         public static Colour Aqua = Colour.FromRgb(0, 255, 255);
00244 /// <summary>
00245 /// Cyan #00FFFF
00246 /// </summary>
00247         public static Colour Cyan = Colour.FromRgb(0, 255, 255);
00248 /// <summary>
00249 /// MidnightBlue #191970
00250 /// </summary>
00251         public static Colour MidnightBlue = Colour.FromRgb(25, 25, 112);
00252 /// <summary>
00253 /// DodgerBlue #1E90FF
00254 /// </summary>
00255         public static Colour DodgerBlue = Colour.FromRgb(30, 144, 255);
00256 /// <summary>
00257 /// LightSeaGreen #20B2AA
00258 /// </summary>
00259         public static Colour LightSeaGreen = Colour.FromRgb(32, 178, 170);
00260 /// <summary>
00261 /// ForestGreen #228B22
00262 /// </summary>
00263         public static Colour ForestGreen = Colour.FromRgb(34, 139, 34);
00264 /// <summary>
00265 /// SeaGreen #2E8B57
00266 /// </summary>
00267         public static Colour SeaGreen = Colour.FromRgb(46, 139, 87);
00268 /// <summary>
00269 /// DarkSlateGray #2F4F4F
00270 /// </summary>
00271         public static Colour DarkSlateGray = Colour.FromRgb(47, 79, 79);
00272 /// <summary>
00273 /// DarkSlateGrey #2F4F4F
```

```
00274 /// </summary>
00275         public static Colour DarkSlateGrey = Colour.FromRgb(47, 79, 79);
00276 /// <summary>
00277 /// LimeGreen #32CD32
00278 /// </summary>
00279         public static Colour LimeGreen = Colour.FromRgb(50, 205, 50);
00280 /// <summary>
00281 /// MediumSeaGreen #3CB371
00282 /// </summary>
00283         public static Colour MediumSeaGreen = Colour.FromRgb(60, 179, 113);
00284 /// <summary>
00285 /// Turquoise #40E0D0
00286 /// </summary>
00287         public static Colour Turquoise = Colour.FromRgb(64, 224, 208);
00288 /// <summary>
00289 /// RoyalBlue #4169E1
00290 /// </summary>
00291         public static Colour RoyalBlue = Colour.FromRgb(65, 105, 225);
00292 /// <summary>
00293 /// SteelBlue #4682B4
00294 /// </summary>
00295         public static Colour SteelBlue = Colour.FromRgb(70, 130, 180);
00296 /// <summary>
00297 /// DarkSlateBlue #483D8B
00298 /// </summary>
00299         public static Colour DarkSlateBlue = Colour.FromRgb(72, 61, 139);
00300 /// <summary>
00301 /// MediumTurquoise #48D1CC
00302 /// </summary>
00303         public static Colour MediumTurquoise = Colour.FromRgb(72, 209, 204);
00304 /// <summary>
00305 /// Indigo #4B0082
00306 /// </summary>
00307         public static Colour Indigo = Colour.FromRgb(75, 0, 130);
00308 /// <summary>
00309 /// DarkOliveGreen #556B2F
00310 /// </summary>
00311         public static Colour DarkOliveGreen = Colour.FromRgb(85, 107, 47);
00312 /// <summary>
00313 /// CadetBlue #5F9EA0
00314 /// </summary>
00315         public static Colour CadetBlue = Colour.FromRgb(95, 158, 160);
00316 /// <summary>
00317 /// CornflowerBlue #6495ED
00318 /// </summary>
00319         public static Colour CornflowerBlue = Colour.FromRgb(100, 149, 237);
00320 /// <summary>
00321 /// RebeccaPurple #663399
00322 /// </summary>
00323         public static Colour RebeccaPurple = Colour.FromRgb(102, 51, 153);
00324 /// <summary>
00325 /// MediumAquaMarine #66CDAA
00326 /// </summary>
00327         public static Colour MediumAquaMarine = Colour.FromRgb(102, 205, 170);
00328 /// <summary>
00329 /// DimGray #696969
00330 /// </summary>
00331         public static Colour DimGray = Colour.FromRgb(105, 105, 105);
00332 /// <summary>
00333 /// DimGrey #696969
00334 /// </summary>
00335         public static Colour DimGrey = Colour.FromRgb(105, 105, 105);
00336 /// <summary>
00337 /// SlateBlue #6A5ACD
00338 /// </summary>
00339         public static Colour SlateBlue = Colour.FromRgb(106, 90, 205);
00340 /// <summary>
00341 /// OliveDrab #6B8E23
00342 /// </summary>
00343         public static Colour OliveDrab = Colour.FromRgb(107, 142, 35);
00344 /// <summary>
00345 /// SlateGray #708090
00346 /// </summary>
00347         public static Colour SlateGray = Colour.FromRgb(112, 128, 144);
00348 /// <summary>
00349 /// SlateGrey #708090
00350 /// </summary>
00351         public static Colour SlateGrey = Colour.FromRgb(112, 128, 144);
00352 /// <summary>
00353 /// LightSlateGray #778899
00354 /// </summary>
00355         public static Colour LightSlateGray = Colour.FromRgb(119, 136, 153);
00356 /// <summary>
00357 /// LightSlateGrey #778899
00358 /// </summary>
00359         public static Colour LightSlateGrey = Colour.FromRgb(119, 136, 153);
00360 /// <summary>
```

```
00361 /// MediumSlateBlue #7B68EE
00362 /// </summary>
00363         public static Colour MediumSlateBlue = Colour.FromRgb(123, 104, 238);
00364 /// <summary>
00365 /// LawnGreen #7CFC00
00366 /// </summary>
00367         public static Colour LawnGreen = Colour.FromRgb(124, 252, 0);
00368 /// <summary>
00369 /// Chartreuse #7FFF00
00370 /// </summary>
00371         public static Colour Chartreuse = Colour.FromRgb(127, 255, 0);
00372 /// <summary>
00373 /// Aquamarine #7FFFD4
00374 /// </summary>
00375         public static Colour Aquamarine = Colour.FromRgb(127, 255, 212);
00376 /// <summary>
00377 /// Maroon #800000
00378 /// </summary>
00379         public static Colour Maroon = Colour.FromRgb(128, 0, 0);
00380 /// <summary>
00381 /// Purple #800080
00382 /// </summary>
00383         public static Colour Purple = Colour.FromRgb(128, 0, 128);
00384 /// <summary>
00385 /// Olive #808000
00386 /// </summary>
00387         public static Colour Olive = Colour.FromRgb(128, 128, 0);
00388 /// <summary>
00389 /// Gray #808080
00390 /// </summary>
00391         public static Colour Gray = Colour.FromRgb(128, 128, 128);
00392 /// <summary>
00393 /// Grey #808080
00394 /// </summary>
00395         public static Colour Grey = Colour.FromRgb(128, 128, 128);
00396 /// <summary>
00397 /// SkyBlue #87CEEB
00398 /// </summary>
00399         public static Colour SkyBlue = Colour.FromRgb(135, 206, 235);
00400 /// <summary>
00401 /// LightSkyBlue #87CEFA
00402 /// </summary>
00403         public static Colour LightSkyBlue = Colour.FromRgb(135, 206, 250);
00404 /// <summary>
00405 /// BlueViolet #8A2BE2
00406 /// </summary>
00407         public static Colour BlueViolet = Colour.FromRgb(138, 43, 226);
00408 /// <summary>
00409 /// DarkRed #8B0000
00410 /// </summary>
00411         public static Colour DarkRed = Colour.FromRgb(139, 0, 0);
00412 /// <summary>
00413 /// DarkMagenta #8B008B
00414 /// </summary>
00415         public static Colour DarkMagenta = Colour.FromRgb(139, 0, 139);
00416 /// <summary>
00417 /// SaddleBrown #8B4513
00418 /// </summary>
00419         public static Colour SaddleBrown = Colour.FromRgb(139, 69, 19);
00420 /// <summary>
00421 /// DarkSeaGreen #8FBC8F
00422 /// </summary>
00423         public static Colour DarkSeaGreen = Colour.FromRgb(143, 188, 143);
00424 /// <summary>
00425 /// LightGreen #90EE90
00426 /// </summary>
00427         public static Colour LightGreen = Colour.FromRgb(144, 238, 144);
00428 /// <summary>
00429 /// MediumPurple #9370DB
00430 /// </summary>
00431         public static Colour MediumPurple = Colour.FromRgb(147, 112, 219);
00432 /// <summary>
00433 /// DarkViolet #9400D3
00434 /// </summary>
00435         public static Colour DarkViolet = Colour.FromRgb(148, 0, 211);
00436 /// <summary>
00437 /// PaleGreen #98FB98
00438 /// </summary>
00439         public static Colour PaleGreen = Colour.FromRgb(152, 251, 152);
00440 /// <summary>
00441 /// DarkOrchid #9932CC
00442 /// </summary>
00443         public static Colour DarkOrchid = Colour.FromRgb(153, 50, 204);
00444 /// <summary>
00445 /// YellowGreen #9ACD32
00446 /// </summary>
00447         public static Colour YellowGreen = Colour.FromRgb(154, 205, 50);
```

```
00448 /// <summary>
00449 /// Sienna #A0522D
00450 /// </summary>
00451         public static Colour Sienna = Colour.FromRgb(160, 82, 45);
00452 /// <summary>
00453 /// Brown #A52A2A
00454 /// </summary>
00455         public static Colour Brown = Colour.FromRgb(165, 42, 42);
00456 /// <summary>
00457 /// DarkGray #A9A9A9
00458 /// </summary>
00459         public static Colour DarkGray = Colour.FromRgb(169, 169, 169);
00460 /// <summary>
00461 /// DarkGrey #A9A9A9
00462 /// </summary>
00463         public static Colour DarkGrey = Colour.FromRgb(169, 169, 169);
00464 /// <summary>
00465 /// LightBlue #ADD8E6
00466 /// </summary>
00467         public static Colour LightBlue = Colour.FromRgb(173, 216, 230);
00468 /// <summary>
00469 /// GreenYellow #ADFF2F
00470 /// </summary>
00471         public static Colour GreenYellow = Colour.FromRgb(173, 255, 47);
00472 /// <summary>
00473 /// PaleTurquoise #AFEEEE
00474 /// </summary>
00475         public static Colour PaleTurquoise = Colour.FromRgb(175, 238, 238);
00476 /// <summary>
00477 /// LightSteelBlue #B0C4DE
00478 /// </summary>
00479         public static Colour LightSteelBlue = Colour.FromRgb(176, 196, 222);
00480 /// <summary>
00481 /// PowderBlue #B0E0E6
00482 /// </summary>
00483         public static Colour PowderBlue = Colour.FromRgb(176, 224, 230);
00484 /// <summary>
00485 /// FireBrick #B22222
00486 /// </summary>
00487         public static Colour FireBrick = Colour.FromRgb(178, 34, 34);
00488 /// <summary>
00489 /// DarkGoldenRod #B8860B
00490 /// </summary>
00491         public static Colour DarkGoldenRod = Colour.FromRgb(184, 134, 11);
00492 /// <summary>
00493 /// MediumOrchid #BA55D3
00494 /// </summary>
00495         public static Colour MediumOrchid = Colour.FromRgb(186, 85, 211);
00496 /// <summary>
00497 /// RosyBrown #BC8F8F
00498 /// </summary>
00499         public static Colour RosyBrown = Colour.FromRgb(188, 143, 143);
00500 /// <summary>
00501 /// DarkKhaki #BDB76B
00502 /// </summary>
00503         public static Colour DarkKhaki = Colour.FromRgb(189, 183, 107);
00504 /// <summary>
00505 /// Silver #C0C0C0
00506 /// </summary>
00507         public static Colour Silver = Colour.FromRgb(192, 192, 192);
00508 /// <summary>
00509 /// MediumVioletRed #C71585
00510 /// </summary>
00511         public static Colour MediumVioletRed = Colour.FromRgb(199, 21, 133);
00512 /// <summary>
00513 /// IndianRed #CD5C5C
00514 /// </summary>
00515         public static Colour IndianRed = Colour.FromRgb(205, 92, 92);
00516 /// <summary>
00517 /// Peru #CD853F
00518 /// </summary>
00519         public static Colour Peru = Colour.FromRgb(205, 133, 63);
00520 /// <summary>
00521 /// Chocolate #D2691E
00522 /// </summary>
00523         public static Colour Chocolate = Colour.FromRgb(210, 105, 30);
00524 /// <summary>
00525 /// Tan #D2B48C
00526 /// </summary>
00527         public static Colour Tan = Colour.FromRgb(210, 180, 140);
00528 /// <summary>
00529 /// LightGray #D3D3D3
00530 /// </summary>
00531         public static Colour LightGray = Colour.FromRgb(211, 211, 211);
00532 /// <summary>
00533 /// LightGrey #D3D3D3
00534 /// </summary>
```

```
00535         public static Colour LightGrey = Colour.FromRgb(211, 211, 211);
00536 /// <summary>
00537 /// Thistle #D8BFD8
00538 /// </summary>
00539         public static Colour Thistle = Colour.FromRgb(216, 191, 216);
00540 /// <summary>
00541 /// Orchid #DA70D6
00542 /// </summary>
00543         public static Colour Orchid = Colour.FromRgb(218, 112, 214);
00544 /// <summary>
00545 /// GoldenRod #DAA520
00546 /// </summary>
00547         public static Colour GoldenRod = Colour.FromRgb(218, 165, 32);
00548 /// <summary>
00549 /// PaleVioletRed #DB7093
00550 /// </summary>
00551         public static Colour PaleVioletRed = Colour.FromRgb(219, 112, 147);
00552 /// <summary>
00553 /// Crimson #DC143C
00554 /// </summary>
00555         public static Colour Crimson = Colour.FromRgb(220, 20, 60);
00556 /// <summary>
00557 /// Gainsboro #DCDCDC
00558 /// </summary>
00559         public static Colour Gainsboro = Colour.FromRgb(220, 220, 220);
00560 /// <summary>
00561 /// Plum #DDA0DD
00562 /// </summary>
00563         public static Colour Plum = Colour.FromRgb(221, 160, 221);
00564 /// <summary>
00565 /// BurlyWood #DEB887
00566 /// </summary>
00567         public static Colour BurlyWood = Colour.FromRgb(222, 184, 135);
00568 /// <summary>
00569 /// LightCyan #E0FFFF
00570 /// </summary>
00571         public static Colour LightCyan = Colour.FromRgb(224, 255, 255);
00572 /// <summary>
00573 /// Lavender #E6E6FA
00574 /// </summary>
00575         public static Colour Lavender = Colour.FromRgb(230, 230, 250);
00576 /// <summary>
00577 /// DarkSalmon #E9967A
00578 /// </summary>
00579         public static Colour DarkSalmon = Colour.FromRgb(233, 150, 122);
00580 /// <summary>
00581 /// Violet #EE82EE
00582 /// </summary>
00583         public static Colour Violet = Colour.FromRgb(238, 130, 238);
00584 /// <summary>
00585 /// PaleGoldenRod #EEE8AA
00586 /// </summary>
00587         public static Colour PaleGoldenRod = Colour.FromRgb(238, 232, 170);
00588 /// <summary>
00589 /// LightCoral #F08080
00590 /// </summary>
00591         public static Colour LightCoral = Colour.FromRgb(240, 128, 128);
00592 /// <summary>
00593 /// Khaki #F0E68C
00594 /// </summary>
00595         public static Colour Khaki = Colour.FromRgb(240, 230, 140);
00596 /// <summary>
00597 /// AliceBlue #F0F8FF
00598 /// </summary>
00599         public static Colour AliceBlue = Colour.FromRgb(240, 248, 255);
00600 /// <summary>
00601 /// HoneyDew #F0FFF0
00602 /// </summary>
00603         public static Colour HoneyDew = Colour.FromRgb(240, 255, 240);
00604 /// <summary>
00605 /// Azure #F0FFFF
00606 /// </summary>
00607         public static Colour Azure = Colour.FromRgb(240, 255, 255);
00608 /// <summary>
00609 /// SandyBrown #F4A460
00610 /// </summary>
00611         public static Colour SandyBrown = Colour.FromRgb(244, 164, 96);
00612 /// <summary>
00613 /// Wheat #F5DEB3
00614 /// </summary>
00615         public static Colour Wheat = Colour.FromRgb(245, 222, 179);
00616 /// <summary>
00617 /// Beige #F5F5DC
00618 /// </summary>
00619         public static Colour Beige = Colour.FromRgb(245, 245, 220);
00620 /// <summary>
00621 /// WhiteSmoke #F5F5F5
```

```
00622 /// </summary>
00623         public static Colour WhiteSmoke = Colour.FromRgb(245, 245, 245);
00624 /// <summary>
00625 /// MintCream #F5FFFA
00626 /// </summary>
00627         public static Colour MintCream = Colour.FromRgb(245, 255, 250);
00628 /// <summary>
00629 /// GhostWhite #F8F8FF
00630 /// </summary>
00631         public static Colour GhostWhite = Colour.FromRgb(248, 248, 255);
00632 /// <summary>
00633 /// Salmon #FA8072
00634 /// </summary>
00635         public static Colour Salmon = Colour.FromRgb(250, 128, 114);
00636 /// <summary>
00637 /// AntiqueWhite #FAEBD7
00638 /// </summary>
00639         public static Colour AntiqueWhite = Colour.FromRgb(250, 235, 215);
00640 /// <summary>
00641 /// Linen #FAF0E6
00642 /// </summary>
00643         public static Colour Linen = Colour.FromRgb(250, 240, 230);
00644 /// <summary>
00645 /// LightGoldenRodYellow #FAFAD2
00646 /// </summary>
00647         public static Colour LightGoldenRodYellow = Colour.FromRgb(250, 250, 210);
00648 /// <summary>
00649 /// OldLace #FDF5E6
00650 /// </summary>
00651         public static Colour OldLace = Colour.FromRgb(253, 245, 230);
00652 /// <summary>
00653 /// Red #FF0000
00654 /// </summary>
00655         public static Colour Red = Colour.FromRgb(255, 0, 0);
00656 /// <summary>
00657 /// Fuchsia #FF00FF
00658 /// </summary>
00659         public static Colour Fuchsia = Colour.FromRgb(255, 0, 255);
00660 /// <summary>
00661 /// Magenta #FF00FF
00662 /// </summary>
00663         public static Colour Magenta = Colour.FromRgb(255, 0, 255);
00664 /// <summary>
00665 /// DeepPink #FF1493
00666 /// </summary>
00667         public static Colour DeepPink = Colour.FromRgb(255, 20, 147);
00668 /// <summary>
00669 /// OrangeRed #FF4500
00670 /// </summary>
00671         public static Colour OrangeRed = Colour.FromRgb(255, 69, 0);
00672 /// <summary>
00673 /// Tomato #FF6347
00674 /// </summary>
00675         public static Colour Tomato = Colour.FromRgb(255, 99, 71);
00676 /// <summary>
00677 /// HotPink #FF69B4
00678 /// </summary>
00679         public static Colour HotPink = Colour.FromRgb(255, 105, 180);
00680 /// <summary>
00681 /// Coral #FF7F50
00682 /// </summary>
00683         public static Colour Coral = Colour.FromRgb(255, 127, 80);
00684 /// <summary>
00685 /// DarkOrange #FF8C00
00686 /// </summary>
00687         public static Colour DarkOrange = Colour.FromRgb(255, 140, 0);
00688 /// <summary>
00689 /// LightSalmon #FFA07A
00690 /// </summary>
00691         public static Colour LightSalmon = Colour.FromRgb(255, 160, 122);
00692 /// <summary>
00693 /// Orange #FFA500
00694 /// </summary>
00695         public static Colour Orange = Colour.FromRgb(255, 165, 0);
00696 /// <summary>
00697 /// LightPink #FFB6C1
00698 /// </summary>
00699         public static Colour LightPink = Colour.FromRgb(255, 182, 193);
00700 /// <summary>
00701 /// Pink #FFC0CB
00702 /// </summary>
00703         public static Colour Pink = Colour.FromRgb(255, 192, 203);
00704 /// <summary>
00705 /// Gold #FFD700
00706 /// </summary>
00707         public static Colour Gold = Colour.FromRgb(255, 215, 0);
00708 /// <summary>
```

```
00709 /// PeachPuff #FFDAB9
00710 /// </summary>
00711         public static Colour PeachPuff = Colour.FromRgb(255, 218, 185);
00712 /// <summary>
00713 /// NavajoWhite #FFDEAD
00714 /// </summary>
00715         public static Colour NavajoWhite = Colour.FromRgb(255, 222, 173);
00716 /// <summary>
00717 /// Moccasin #FFE4B5
00718 /// </summary>
00719         public static Colour Moccasin = Colour.FromRgb(255, 228, 181);
00720 /// <summary>
00721 /// Bisque #FFE4C4
00722 /// </summary>
00723         public static Colour Bisque = Colour.FromRgb(255, 228, 196);
00724 /// <summary>
00725 /// MistyRose #FFE4E1
00726 /// </summary>
00727         public static Colour MistyRose = Colour.FromRgb(255, 228, 225);
00728 /// <summary>
00729 /// BlanchedAlmond #FFEBCD
00730 /// </summary>
00731         public static Colour BlanchedAlmond = Colour.FromRgb(255, 235, 205);
00732 /// <summary>
00733 /// PapayaWhip #FFEFD5
00734 /// </summary>
00735         public static Colour PapayaWhip = Colour.FromRgb(255, 239, 213);
00736 /// <summary>
00737 /// LavenderBlush #FFF0F5
00738 /// </summary>
00739         public static Colour LavenderBlush = Colour.FromRgb(255, 240, 245);
00740 /// <summary>
00741 /// SeaShell #FFF5EE
00742 /// </summary>
00743         public static Colour SeaShell = Colour.FromRgb(255, 245, 238);
00744 /// <summary>
00745 /// Cornsilk #FFF8DC
00746 /// </summary>
00747         public static Colour Cornsilk = Colour.FromRgb(255, 248, 220);
00748 /// <summary>
00749 /// LemonChiffon #FFFACD
00750 /// </summary>
00751         public static Colour LemonChiffon = Colour.FromRgb(255, 250, 205);
00752 /// <summary>
00753 /// FloralWhite #FFFAF0
00754 /// </summary>
00755         public static Colour FloralWhite = Colour.FromRgb(255, 250, 240);
00756 /// <summary>
00757 /// Snow #FFFAFA
00758 /// </summary>
00759         public static Colour Snow = Colour.FromRgb(255, 250, 250);
00760 /// <summary>
00761 /// Yellow #FFFF00
00762 /// </summary>
00763         public static Colour Yellow = Colour.FromRgb(255, 255, 0);
00764 /// <summary>
00765 /// LightYellow #FFFFE0
00766 /// </summary>
00767         public static Colour LightYellow = Colour.FromRgb(255, 255, 224);
00768 /// <summary>
00769 /// Ivory #FFFFF0
00770 /// </summary>
00771         public static Colour Ivory = Colour.FromRgb(255, 255, 240);
00772 /// <summary>
00773 /// White #FFFFFF
00774 /// </summary>
00775         public static Colour White = Colour.FromRgb(255, 255, 255);
00776     }
00777 }
```

8.76 TrueType.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Text;
00023
00024 namespace VectSharp
00025 {
00026 /// <summary>
00027 /// Represents a font file in TrueType format.  Reference:  http://stevehanov.ca/blog/?id=143,
      https://developer.apple.com/fonts/TrueType-Reference-Manual/,
      https://docs.microsoft.com/en-us/typography/opentype/spec/
00028 /// </summary>
00029     public class TrueTypeFile
00030     {
00031         private static readonly Dictionary<string, TrueTypeFile> FontCache = new Dictionary<string,
      TrueTypeFile>();
00032         private static readonly Dictionary<Stream, TrueTypeFile> StreamFontCache = new
      Dictionary<Stream, TrueTypeFile>();
00033         private static object FontCacheLock = new object();
00034         internal uint ScalarType { get; }
00035         internal ushort NumTables { get; }
00036         internal ushort SearchRange { get; }
00037         internal ushort EntrySelector { get; }
00038         internal ushort RangeShift { get; }
00039         internal Dictionary<string, TrueTypeTableOffset> TableOffsets { get; } = new
      Dictionary<string, TrueTypeTableOffset>();
00040         internal Dictionary<string, ITrueTypeTable> Tables { get; } = new Dictionary<string,
      ITrueTypeTable>();
00041
00042
00043 /// <summary>
00044 /// A stream pointing to the TrueType file source (either on disk or in memory).  Never dispose this
      stream directly; if you really need to, call <see cref="Destroy"/> instead.
00045 /// </summary>
00046         public Stream FontStream { get; }
00047
00048 /// <summary>
00049 /// Remove this TrueType file from the cache, clear the tables and release the <see
      cref="FontStream"/>.
00050 /// Only call this when the actual file that was used to create this object needs to be changed!
00051 /// </summary>
00052         public void Destroy()
00053         {
00054             string keyString = null;
00055
00056
00057             lock (FontCacheLock)
00058             {
00059                 foreach (KeyValuePair<string, TrueTypeFile> kvp in FontCache)
00060                 {
00061                     if (kvp.Value == this)
00062                     {
00063                         keyString = kvp.Key;
00064                         break;
00065                     }
00066                 }
00067
00068
00069                 if (!string.IsNullOrEmpty(keyString))
00070                 {
00071                     FontCache.Remove(keyString);
00072                 }
00073
00074                 Stream keyStream = null;
00075
00076                 foreach (KeyValuePair<Stream, TrueTypeFile> kvp in StreamFontCache)
00077                 {
00078                     if (kvp.Value == this)
00079                     {
00080                         keyStream = kvp.Key;
00081                     }
00082                 }
00083
00084                 if (keyStream != null)
00085                 {
00086                     StreamFontCache.Remove(keyStream);
00087                 }
00088             }
00089
00090             this.FontStream.Dispose();
```

```
00091             this.TableOffsets.Clear();
00092             this.Tables.Clear();
00093         }
00094 
00095         internal TrueTypeFile(Dictionary<string, ITrueTypeTable> tables)
00096         {
00097             this.Tables = tables;
00098 
00099             this.ScalarType = (uint)65536;
00100             this.NumTables = (ushort)tables.Count;
00101             this.EntrySelector = (ushort)Math.Floor(Math.Log(tables.Count, 2));
00102             this.SearchRange = (ushort)(16 * (1 << this.EntrySelector));
00103             this.RangeShift = (ushort)(tables.Count * 16 - this.SearchRange);
00104 
00105             uint offset = 12 + 16 * (uint)tables.Count;
00106 
00107             uint fontChecksum = this.ScalarType;
00108             fontChecksum += ((uint)this.NumTables << 16) + (uint)this.SearchRange;
00109             fontChecksum += ((uint)this.EntrySelector << 16) + (uint)this.RangeShift;
00110 
00111             //For some reason, OpenType Sanitizer (https://github.com/khaledhosny/ots), integrated
    into Chromium and Firefox, complains if the tables are ordered differently.
00112             List<string> tableOrdering = new List<string>()
00113             {
00114                 "OS/2", "cmap", "glyf", "head", "hhea", "hmtx", "loca", "maxp", "name", "post"
00115             };
00116 
00117             Dictionary<string, byte[]> allBytes = new Dictionary<string, byte[]>();
00118 
00119             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
    tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ?  ind :  tableOrdering.Count) ascending select el)
00120             {
00121                 if (kvp.Key == "head")
00122                 {
00123                     ((TrueTypeHeadTable)kvp.Value).ChecksumAdjustment = 0;
00124                 }
00125 
00126                 byte[] tableBytes = kvp.Value.GetBytes();
00127                 allBytes.Add(kvp.Key, tableBytes);
00128                 int length = tableBytes.Length;
00129 
00130                 uint checksum = 0;
00131 
00132                 for (int i = 0; i < tableBytes.Length; i += 4)
00133                 {
00134                     byte b1 = tableBytes[i];
00135                     byte b2 = (i + 1 < tableBytes.Length) ?  tableBytes[i + 1] :  (byte)0;
00136                     byte b3 = (i + 2 < tableBytes.Length) ?  tableBytes[i + 2] :  (byte)0;
00137                     byte b4 = (i + 3 < tableBytes.Length) ?  tableBytes[i + 3] :  (byte)0;
00138 
00139                     checksum += (uint)((b1 << 24) + (b2 << 16) + (b3 << 8) + b4);
00140                 }
00141 
00142                 TableOffsets.Add(kvp.Key, new TrueTypeTableOffset(checksum, offset, (uint)length));
00143 
00144                 fontChecksum += checksum;
00145 
00146                 fontChecksum += (uint)(((byte)kvp.Key[0] << 24) + ((byte)kvp.Key[1] << 16) +
    ((byte)kvp.Key[2] << 8) + (byte)kvp.Key[3]);
00147                 fontChecksum += offset;
00148                 fontChecksum += checksum;
00149                 fontChecksum += (uint)length;
00150 
00151                 offset += (uint)length;
00152 
00153                 switch (length % 4)
00154                 {
00155                     case 1:
00156                         offset += 3;
00157                         break;
00158                     case 2:
00159                         offset += 2;
00160                         break;
00161                     case 3:
00162                         offset++;
00163                         break;
00164                 }
00165             }
00166 
00167             ((TrueTypeHeadTable)this.Tables["head"]).ChecksumAdjustment = 0xB1B0AFBA - fontChecksum;
00168 
00169 
00170             FontStream = new MemoryStream();
00171             FontStream.WriteUInt(this.ScalarType);
00172             FontStream.WriteUShort(this.NumTables);
00173             FontStream.WriteUShort(this.SearchRange);
00174             FontStream.WriteUShort(this.EntrySelector);
```

```
00175             FontStream.WriteUShort(this.RangeShift);
00176
00177             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
      tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ?  ind :  tableOrdering.Count) ascending select el)
00178             {
00179                 FontStream.Write(Encoding.ASCII.GetBytes(kvp.Key), 0, 4);
00180                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Checksum);
00181                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Offset);
00182                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Length);
00183             }
00184
00185             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
      tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ?  ind :  tableOrdering.Count) ascending select el)
00186             {
00187                 FontStream.Write(allBytes[kvp.Key], 0, allBytes[kvp.Key].Length);
00188                 switch (allBytes[kvp.Key].Length % 4)
00189                 {
00190                     case 1:
00191                         FontStream.WriteByte(0);
00192                         FontStream.WriteByte(0);
00193                         FontStream.WriteByte(0);
00194                         break;
00195                     case 2:
00196                         FontStream.WriteByte(0);
00197                         FontStream.WriteByte(0);
00198                         break;
00199                     case 3:
00200                         FontStream.WriteByte(0);
00201                         break;
00202                 }
00203             }
00204
00205             FontStream.Seek(0, SeekOrigin.Begin);
00206
00207             BuildGlyphCache();
00208         }
00209
00210         internal static TrueTypeFile CreateTrueTypeFile(string fileName)
00211         {
00212             lock (FontCacheLock)
00213             {
00214                 if (!FontCache.ContainsKey(fileName))
00215                 {
00216                     FontCache.Add(fileName, new TrueTypeFile(fileName));
00217                 }
00218
00219                 return FontCache[fileName];
00220             }
00221         }
00222
00223         internal static TrueTypeFile CreateTrueTypeFile(Stream fontStream)
00224         {
00225             lock (FontCacheLock)
00226             {
00227                 if (!StreamFontCache.ContainsKey(fontStream))
00228                 {
00229                     StreamFontCache.Add(fontStream, new TrueTypeFile(fontStream));
00230                 }
00231
00232                 return StreamFontCache[fontStream];
00233             }
00234         }
00235
00236         private TrueTypeFile(string fileName) :  this(new FileStream(fileName, FileMode.Open,
      FileAccess.Read, FileShare.Read))
00237         {
00238
00239         }
00240
00241         private TrueTypeFile(Stream fs)
00242         {
00243             FontStream = fs;
00244             this.ScalarType = fs.ReadUInt();
00245             this.NumTables = fs.ReadUShort();
00246             this.SearchRange = fs.ReadUShort();
00247             this.EntrySelector = fs.ReadUShort();
00248             this.RangeShift = fs.ReadUShort();
00249
00250             for (int i = 0; i < NumTables; i++)
00251             {
00252                 this.TableOffsets.Add(fs.ReadString(4), new TrueTypeTableOffset(fs.ReadUInt(),
      fs.ReadUInt(), fs.ReadUInt()));
00253             }
00254
00255             fs.Seek(this.TableOffsets["head"].Offset, SeekOrigin.Begin);
00256
00257             Tables.Add("head", new TrueTypeHeadTable()
```

```
00258             {
00259                 Version = fs.ReadFixed(),
00260                 FontRevision = fs.ReadFixed(),
00261                 ChecksumAdjustment = fs.ReadUInt(),
00262                 MagicNumber = fs.ReadUInt(),
00263                 Flags = fs.ReadUShort(),
00264                 UnitsPerEm = fs.ReadUShort(),
00265                 Created = fs.ReadDate(),
00266                 Modified = fs.ReadDate(),
00267                 XMin = fs.ReadShort(),
00268                 YMin = fs.ReadShort(),
00269                 XMax = fs.ReadShort(),
00270                 YMax = fs.ReadShort(),
00271                 MacStyle = fs.ReadUShort(),
00272                 LowestRecPPEM = fs.ReadUShort(),
00273                 FontDirectionInt = fs.ReadShort(),
00274                 IndexToLocFormat = fs.ReadShort(),
00275                 GlyphDataFormat = fs.ReadShort()
00276             });
00277
00278             fs.Seek(this.TableOffsets["hhea"].Offset, SeekOrigin.Begin);
00279
00280             Tables.Add("hhea", new TrueTypeHHeaTable()
00281             {
00282                 Version = fs.ReadFixed(),
00283                 Ascent = fs.ReadShort(),
00284                 Descent = fs.ReadShort(),
00285                 LineGap = fs.ReadShort(),
00286                 AdvanceWidthMax = fs.ReadUShort(),
00287                 MinLeftSideBearing = fs.ReadShort(),
00288                 MinRightSideBearing = fs.ReadShort(),
00289                 XMaxExtent = fs.ReadShort(),
00290                 CaretSlopeRise = fs.ReadShort(),
00291                 CaretSlopeRun = fs.ReadShort(),
00292                 CaretOffset = fs.ReadShort()
00293             });
00294
00295             fs.ReadShort();
00296             fs.ReadShort();
00297             fs.ReadShort();
00298             fs.ReadShort();
00299
00300             ((TrueTypeHHeaTable)Tables["hhea"]).MetricDataFormat = fs.ReadShort();
00301             ((TrueTypeHHeaTable)Tables["hhea"]).NumOfLongHorMetrics = fs.ReadUShort();
00302
00303             fs.Seek(this.TableOffsets["maxp"].Offset, SeekOrigin.Begin);
00304
00305             Tables.Add("maxp", new TrueTypeMaxpTable()
00306             {
00307                 Version = fs.ReadFixed(),
00308                 NumGlyphs = fs.ReadUShort()
00309             });
00310
00311             if (((TrueTypeMaxpTable)Tables["maxp"]).Version.Bits == 65536)
00312             {
00313                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxPoints = fs.ReadUShort();
00314                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxContours = fs.ReadUShort();
00315                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentPoints = fs.ReadUShort();
00316                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentContours = fs.ReadUShort();
00317                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxZones = fs.ReadUShort();
00318                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxTwilightPoints = fs.ReadUShort();
00319                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxStorage = fs.ReadUShort();
00320                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxFunctionDefs = fs.ReadUShort();
00321                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxInstructionDefs = fs.ReadUShort();
00322                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxStackElements = fs.ReadUShort();
00323                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxSizeOfInstructions = fs.ReadUShort();
00324                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentElements = fs.ReadUShort();
00325                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentDepth = fs.ReadUShort();
00326             }
00327
00328             int totalGlyphs = ((TrueTypeMaxpTable)Tables["maxp"]).NumGlyphs;
00329             int numMetrics = ((TrueTypeHHeaTable)Tables["hhea"]).NumOfLongHorMetrics;
00330
00331             fs.Seek(this.TableOffsets["hmtx"].Offset, SeekOrigin.Begin);
00332             Tables.Add("hmtx", new TrueTypeHmtxTable() { HMetrics = new LongHorFixed[numMetrics],
    LeftSideBearing = new short[totalGlyphs - numMetrics] });
00333
00334             for (int i = 0; i < numMetrics; i++)
00335             {
00336                 ((TrueTypeHmtxTable)Tables["hmtx"]).HMetrics[i] = new
    TrueTypeFile.LongHorFixed(fs.ReadUShort(), fs.ReadShort());
00337             }
00338
00339             for (int i = 0; i < totalGlyphs - numMetrics; i++)
00340             {
00341                 ((TrueTypeHmtxTable)Tables["hmtx"]).LeftSideBearing[i] = fs.ReadShort();
00342             }
```

```
00343
00344             fs.Seek(this.TableOffsets["cmap"].Offset, SeekOrigin.Begin);
00345             Tables.Add("cmap", new TrueTypeCmapTable() { Version = fs.ReadUShort(), NumberSubTables =
      fs.ReadUShort() });
00346
00347             ((TrueTypeCmapTable)Tables["cmap"]).SubTables = new
      CmapSubTable[((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables];
00348             ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables = new
      ICmapTable[((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables];
00349
00350             for (int i = 0; i < ((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables; i++)
00351             {
00352                 ((TrueTypeCmapTable)Tables["cmap"]).SubTables[i] = new
      TrueTypeFile.CmapSubTable(fs.ReadUShort(), fs.ReadUShort(), fs.ReadUInt());
00353             }
00354
00355
00356             for (int i = 0; i < ((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables; i++)
00357             {
00358                 fs.Seek(((TrueTypeCmapTable)Tables["cmap"]).SubTables[i].Offset +
      TableOffsets["cmap"].Offset, SeekOrigin.Begin);
00359
00360                 ushort format = fs.ReadUShort();
00361                 ushort length = fs.ReadUShort();
00362                 ushort language = fs.ReadUShort();
00363
00364                 if (format == 0)
00365                 {
00366                     byte[] glyphIndexArray = new byte[256];
00367                     fs.Read(glyphIndexArray, 0, 256);
00368                     ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables[i] = new CmapTable0(format,
      length, language, glyphIndexArray);
00369                 }
00370                 else if (format == 4)
00371                 {
00372                     ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables[i] = new CmapTable4(format,
      length, language, fs);
00373                 }
00374             }
00375
00376             if (this.TableOffsets.ContainsKey("OS/2"))
00377             {
00378                 fs.Seek(this.TableOffsets["OS/2"].Offset, SeekOrigin.Begin);
00379
00380                 TrueTypeOS2Table os2 = new TrueTypeOS2Table()
00381                 {
00382                     Version = fs.ReadUShort(),
00383                     XAvgCharWidth = fs.ReadShort(),
00384                     UsWeightClass = fs.ReadUShort(),
00385                     UsWidthClass = fs.ReadUShort(),
00386                     FsType = fs.ReadShort(),
00387                     YSubscriptXSize = fs.ReadShort(),
00388                     YSubscriptYSize = fs.ReadShort(),
00389                     YSubscriptXOffset = fs.ReadShort(),
00390                     YSubscriptYOffset = fs.ReadShort(),
00391                     YSuperscriptXSize = fs.ReadShort(),
00392                     YSuperscriptYSize = fs.ReadShort(),
00393                     YSuperscriptXOffset = fs.ReadShort(),
00394                     YSuperscriptYOffset = fs.ReadShort(),
00395                     YStrikeoutSize = fs.ReadShort(),
00396                     YStrikeoutPosition = fs.ReadShort(),
00397                     SFamilyClass = (byte)fs.ReadByte(),
00398                     SFamilySubClass = (byte)fs.ReadByte(),
00399                     Panose = new TrueTypeOS2Table.PANOSE((byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte()),
00400                     UlUnicodeRange = new uint[] { fs.ReadUInt(), fs.ReadUInt(), fs.ReadUInt(),
      fs.ReadUInt() },
00401                     AchVendID = new byte[] { (byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte() },
00402                     FsSelection = fs.ReadUShort(),
00403                     FsFirstCharIndex = fs.ReadUShort(),
00404                     FsLastCharIndex = fs.ReadUShort()
00405                 };
00406
00407                 if (os2.Version > 0 || os2.Version == 0 && this.TableOffsets["OS/2"].Length > 68)
00408                 {
00409                     os2.STypoAscender = fs.ReadShort();
00410                     os2.STypoDescender = fs.ReadShort();
00411                     os2.STypoLineGap = fs.ReadShort();
00412                     os2.UsWinAscent = fs.ReadUShort();
00413                     os2.UsWinDescent = fs.ReadUShort();
00414
00415                     if (os2.Version >= 1)
00416                     {
00417                         os2.UlCodePageRange = new uint[] { fs.ReadUInt(), fs.ReadUInt() };
00418
```

```
00419                         if (os2.Version >= 2)
00420                         {
00421                             os2.SxHeight = fs.ReadShort();
00422                             os2.SCapHeight = fs.ReadShort();
00423                             os2.UsDefaultChar = fs.ReadUShort();
00424                             os2.UsBreakChar = fs.ReadUShort();
00425                             os2.UsMaxContext = fs.ReadUShort();
00426 
00427                             if (os2.Version >= 5)
00428                             {
00429                                 os2.UsLowerOpticalPointSize = fs.ReadUShort();
00430                                 os2.UsUpperOpticalPointSize = fs.ReadUShort();
00431                             }
00432                         }
00433                     }
00434                 }
00435 
00436 
00437                 Tables.Add("OS/2", os2);
00438             }
00439 
00440             if (this.TableOffsets.ContainsKey("post"))
00441             {
00442                 fs.Seek(this.TableOffsets["post"].Offset, SeekOrigin.Begin);
00443 
00444                 TrueTypePostTable post = new TrueTypePostTable()
00445                 {
00446                     Version = fs.ReadFixed(),
00447                     ItalicAngle = fs.ReadFixed(),
00448                     UnderlinePosition = fs.ReadShort(),
00449                     UnderlineThickness = fs.ReadShort(),
00450                     IsFixedPitch = fs.ReadUInt(),
00451                     MinMemType42 = fs.ReadUInt(),
00452                     MaxMemType42 = fs.ReadUInt(),
00453                     MinMemType1 = fs.ReadUInt(),
00454                     MaxMemType1 = fs.ReadUInt()
00455                 };
00456 
00457                 if (post.Version.Bits == 0x00020000)
00458                 {
00459                     post.NumGlyphs = fs.ReadUShort();
00460                     post.GlyphNameIndex = new ushort[post.NumGlyphs];
00461 
00462                     int numberNewGlyphs = 0;
00463 
00464                     for (int i = 0; i < post.NumGlyphs; i++)
00465                     {
00466                         post.GlyphNameIndex[i] = fs.ReadUShort();
00467                         if (post.GlyphNameIndex[i] >= 258)
00468                         {
00469                             numberNewGlyphs++;
00470                         }
00471                     }
00472 
00473                     post.Names = new string[numberNewGlyphs];
00474 
00475                     for (int i = 0; i < numberNewGlyphs; i++)
00476                     {
00477                         post.Names[i] = fs.ReadPascalString();
00478                     }
00479                 }
00480                 Tables.Add("post", post);
00481             }
00482 
00483             if (this.TableOffsets.ContainsKey("name"))
00484             {
00485                 fs.Seek(this.TableOffsets["name"].Offset, SeekOrigin.Begin);
00486                 Tables.Add("name", new TrueTypeNameTable(this.TableOffsets["name"].Offset, fs));
00487             }
00488 
00489             fs.Seek(this.TableOffsets["loca"].Offset, SeekOrigin.Begin);
00490             Tables.Add("loca", new TrueTypeLocaTable(fs,
      ((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs,
      ((TrueTypeHeadTable)this.Tables["head"]).IndexToLocFormat == 0));
00491 
00492             if (this.TableOffsets.ContainsKey("cvt "))
00493             {
00494                 fs.Seek(this.TableOffsets["cvt "].Offset, SeekOrigin.Begin);
00495                 Tables.Add("cvt ", new TrueTypeRawTable(fs, this.TableOffsets["cvt "].Length));
00496             }
00497 
00498             if (this.TableOffsets.ContainsKey("prep"))
00499             {
00500                 fs.Seek(this.TableOffsets["prep"].Offset, SeekOrigin.Begin);
00501                 Tables.Add("prep", new TrueTypeRawTable(fs, this.TableOffsets["prep"].Length));
00502             }
00503 
```

```
00504             if (this.TableOffsets.ContainsKey("fpgm"))
00505             {
00506                 fs.Seek(this.TableOffsets["fpgm"].Offset, SeekOrigin.Begin);
00507                 Tables.Add("fpgm", new TrueTypeRawTable(fs, this.TableOffsets["fpgm"].Length));
00508             }
00509 
00510             if (this.TableOffsets.ContainsKey("GPOS"))
00511             {
00512                 fs.Seek(this.TableOffsets["GPOS"].Offset, SeekOrigin.Begin);
00513                 Tables.Add("GPOS", new TrueTypeGPOSTable(fs));
00514             }
00515 
00516             Glyph[] glyphs = new Glyph[((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs];
00517 
00518             for (int i = 0; i < ((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs; i++)
00519             {
00520                 uint offset = ((TrueTypeLocaTable)Tables["loca"]).GetOffset(i);
00521                 uint length = ((TrueTypeLocaTable)Tables["loca"]).Lengths[i];
00522 
00523                 if (length > 0)
00524                 {
00525                     fs.Seek(this.TableOffsets["glyf"].Offset + offset, SeekOrigin.Begin);
00526                     glyphs[i] = Glyph.Parse(fs);
00527                 }
00528                 else
00529                 {
00530                     glyphs[i] = new EmptyGlyph();
00531                 }
00532 
00533             }
00534 
00535             Tables.Add("glyf", new TrueTypeGlyfTable() { Glyphs = glyphs });
00536 
00537             BuildGlyphCache();
00538         }
00539 
00540 /// <summary>
00541 /// Represets a  TrueType name.
00542 /// </summary>
00543         public struct TrueTypeName
00544         {
00545 /// <summary>
00546 /// Describes a kind of TrueType name.
00547 /// </summary>
00548             public enum NameIdentifier
00549             {
00550 /// <summary>
00551 /// A font family name.
00552 /// </summary>
00553                 FontFamily = 1,
00554 
00555 /// <summary>
00556 /// A full name for a font.
00557 /// </summary>
00558                 FullName = 4,
00559 
00560 /// <summary>
00561 /// A PostScript name for a font.
00562 /// </summary>
00563                 PostScriptName = 6,
00564 
00565 /// <summary>
00566 /// The preferred font family name for the font.
00567 /// </summary>
00568                 PreferredFamily = 16
00569             }
00570 
00571 /// <summary>
00572 /// The kind of name represented by this struct.
00573 /// </summary>
00574             public NameIdentifier NameId;
00575 
00576 /// <summary>
00577 /// The name represented by this struct.
00578 /// </summary>
00579             public string Name;
00580 
00581             internal TrueTypeName(NameIdentifier identifier, string name)
00582             {
00583                 this.NameId = identifier;
00584                 this.Name = name;
00585             }
00586         }
00587 
00588 /// <summary>
00589 /// Determines whether the file is a valid TrueType file and retrieves the list of names defined in
      it.
```

```
00590 /// </summary>
00591 /// <param name="file">The TrueType file.</param>
00592 /// <param name="names">The list of names that will be returned.  These will only include names with
      TrueType identifiers 1, 4, 6, or 16.</param>
00593 /// <returns>If an error occurred while opening the file, or the file does not contain a valid
      TrueType file, <see langword="false"/>.  If the TrueType file appears valid but does not contain a
      name table,
00594 /// <see langword="true"/>, but <paramref name="names"/> will be <see langword="null"/>.  Otherwise,
      <see langword="true"/> and <paramref name="names"/> will not be null
00595 /// (but it might still be empty).</returns>
00596         public static bool GetNames(string file, out List<TrueTypeName> names)
00597         {
00598             try
00599             {
00600                 using (FileStream fs = File.Open(file, FileMode.Open, FileAccess.Read,
      FileShare.Read))
00601                 {
00602                     return GetNames(fs, out names);
00603                 }
00604             }
00605             catch
00606             {
00607                 names = null;
00608                 return false;
00609             }
00610         }
00611
00612 /// <summary>
00613 /// Determines whether the stream contains a valid TrueType file and retrieves the list of names
      defined in it.
00614 /// </summary>
00615 /// <param name="fs">The stream containing the TrueType file.</param>
00616 /// <param name="names">The list of names that will be returned.  These will only include names with
      TrueType identifiers 1, 4, 6, or 16.</param>
00617 /// <returns>If the stream does not contain a valid TrueType file, <see langword="false"/>.  If the
      TrueType file appears valid but does not contain a name table,
00618 /// <see langword="true"/>, but <paramref name="names"/> will be <see langword="null"/>.  Otherwise,
      <see langword="true"/> and <paramref name="names"/> will not be null
00619 /// (but it might still be empty).</returns>
00620         public static bool GetNames(Stream fs, out List<TrueTypeName> names)
00621         {
00622             try
00623             {
00624                 uint ScalarType = fs.ReadUInt();
00625
00626                 if (ScalarType != 0x00010000 && ScalarType != 0x74727565)
00627                 {
00628                     names = null;
00629                     return false;
00630                 }
00631
00632                 int NumTables = fs.ReadUShort();
00633                 fs.ReadUShort();
00634                 fs.ReadUShort();
00635                 fs.ReadUShort();
00636
00637                 TrueTypeNameTable name = null;
00638
00639                 for (int i = 0; i < NumTables; i++)
00640                 {
00641                     string key = fs.ReadString(4);
00642                     TrueTypeTableOffset offset = new TrueTypeTableOffset(fs.ReadUInt(), fs.ReadUInt(),
      fs.ReadUInt());
00643
00644                     if (key == "name")
00645                     {
00646                         fs.Seek(offset.Offset, SeekOrigin.Begin);
00647                         name = new TrueTypeNameTable(offset.Offset, fs);
00648                         break;
00649                     }
00650                 }
00651
00652                 names = null;
00653
00654                 if (name != null)
00655                 {
00656                     names = new List<TrueTypeName>();
00657
00658                     for (int i = 0; i < name.NameRecords.Length; i++)
00659                     {
00660                         if (name.NameRecords[i].NameID == 1 || name.NameRecords[i].NameID == 4 ||
      name.NameRecords[i].NameID == 6 || name.NameRecords[i].NameID == 16)
00661                             {
00662                                 names.Add(new
      TrueTypeName((TrueTypeName.NameIdentifier)name.NameRecords[i].NameID, name.Name[i]));
00663                             }
00664                     }
```

```
00665                 }
00666
00667                 return true;
00668             }
00669             catch
00670             {
00671                 names = null;
00672                 return false;
00673             }
00674         }
00675
00676         private double[] GlyphWidthsCache = new double[256];
00677         private VerticalMetrics[] VerticalMetricsCache = new VerticalMetrics[256];
00678         private Bearings[] BearingsCache = new Bearings[256];
00679         private void BuildGlyphCache()
00680         {
00681             for (int i = 0; i < 256; i++)
00682             {
00683                 GlyphWidthsCache[i] = this.Get1000EmGlyphWidth(GetGlyphIndex((char)i));
00684                 VerticalMetricsCache[i] = this.Get1000EmGlyphVerticalMetrics(GetGlyphIndex((char)i));
00685                 BearingsCache[i] = this.Get1000EmGlyphBearings(GetGlyphIndex((char)i));
00686             }
00687         }
00688
00689 /// <summary>
00690 /// Create a subset of the TrueType file, containing only the glyphs for the specified characters.
00691 /// </summary>
00692 /// <param name="charactersToInclude">A string containing the characters for which the glyphs should
      be included.</param>
00693 /// <param name="consolidateAt32">If true, the character map is rearranged so that the included glyphs
      start at the unicode U+0032 control point.</param>
00694 /// <param name="outputEncoding">If <paramref name="consolidateAt32"/> is true, entries will be added
      to this dictionary mapping the original characters to the new map (that starts at U+0033).</param>
00695 /// <returns></returns>
00696         public TrueTypeFile SubsetFont(string charactersToInclude, bool consolidateAt32 = false,
      Dictionary<char, char> outputEncoding = null)
00697         {
00698             if (!this.HasCmap4Table())
00699             {
00700                 return this;
00701             }
00702             else
00703             {
00704
00705                 TrueTypeHeadTable head = (TrueTypeHeadTable)this.Tables["head"];
00706
00707                 TrueTypeHHeaTable originalHhea = (TrueTypeHHeaTable)this.Tables["hhea"];
00708                 TrueTypeMaxpTable originalMaxp = (TrueTypeMaxpTable)this.Tables["maxp"];
00709                 TrueTypeCmapTable originalCmap = (TrueTypeCmapTable)this.Tables["cmap"];
00710                 TrueTypeLocaTable originalLoca = (TrueTypeLocaTable)this.Tables["loca"];
00711                 TrueTypeGlyfTable originalGlyf = (TrueTypeGlyfTable)this.Tables["glyf"];
00712
00713                 CmapTable4 cmap4 = null;
00714                 int platformId = -1;
00715                 int platformSpecificID = -1;
00716
00717                 for (int i = 0; i < originalCmap.ActualCmapTables.Length; i++)
00718                 {
00719                     if (originalCmap.ActualCmapTables[i] != null &&
      originalCmap.ActualCmapTables[i].Format == 4 && originalCmap.SubTables[i].PlatformID == 3 &&
      originalCmap.SubTables[i].PlatformSpecificID == 1)
00720                     {
00721                         cmap4 = (CmapTable4)originalCmap.ActualCmapTables[i];
00722                         platformId = originalCmap.SubTables[i].PlatformID;
00723                         platformSpecificID = originalCmap.SubTables[i].PlatformSpecificID;
00724                         break;
00725                     }
00726                 }
00727
00728                 if (cmap4 == null)
00729                 {
00730                     return this;
00731                 }
00732
00733                 List<int> characterCodes = new List<int>();
00734
00735                 for (int i = 0; i < charactersToInclude.Length; i++)
00736                 {
00737                     characterCodes.Add((int)charactersToInclude[i]);
00738                 }
00739
00740                 characterCodes.Sort();
00741
00742                 List<(int, int)> segments = new List<(int, int)>();
00743
00744                 for (int i = 0; i < characterCodes.Count; i++)
00745                 {
```

```
00746                     if (segments.Count > 0 && segments.Last().Item1 + segments.Last().Item2 ==
      characterCodes[i])
00747                     {
00748                         segments[segments.Count - 1] = (segments[segments.Count - 1].Item1,
      segments[segments.Count - 1].Item2 + 1);
00749                     }
00750                     else
00751                     {
00752                         segments.Add((characterCodes[i], 1));
00753                     }
00754                 }
00755 
00756                 segments.Add((cmap4.StartCode.Last(), cmap4.EndCode.Last() - cmap4.StartCode.Last() +
      1));
00757 
00758                 int totalGlyphIndexCount = characterCodes.Count + 1;
00759 
00760                 segments.Sort((a, b) => a.Item1 + a.Item2 - b.Item1 - b.Item2);
00761 
00762                 ushort entrySelector = (ushort)Math.Floor(Math.Log(segments.Count, 2));
00763 
00764 
00765                 CmapTable4 newCmap4 = new CmapTable4()
00766                 {
00767                     Format = 4,
00768                     Length = (ushort)(16 + 8 * segments.Count),
00769                     Language = cmap4.Language,
00770                     SegCountX2 = (ushort)(2 * segments.Count),
00771                     SearchRange = (ushort)(2 * (1 << entrySelector)),
00772                     EntrySelector = entrySelector,
00773                     RangeShift = (ushort)(2 * segments.Count - (2 * (1 << entrySelector))),
00774                     EndCode = new ushort[segments.Count],
00775                     ReservedPad = cmap4.ReservedPad,
00776                     StartCode = new ushort[segments.Count],
00777                     IdDelta = new ushort[segments.Count],
00778                     IdRangeOffset = new ushort[segments.Count],
00779                     GlyphIndexArray = new ushort[0]
00780                 };
00781 
00782                 List<char> includedGlyphs = new List<char>
00783                 {
00784                     (char)0
00785                 };
00786 
00787                 List<ushort> glyphIndexArray = new List<ushort>();
00788 
00789                 if (!consolidateAt32)
00790                 {
00791                     for (int i = 0; i < segments.Count; i++)
00792                     {
00793                         newCmap4.EndCode[i] = (ushort)(segments[i].Item1 + segments[i].Item2 - 1);
00794                         newCmap4.StartCode[i] = (ushort)segments[i].Item1;
00795 
00796                         newCmap4.IdRangeOffset[i] = 0;
00797 
00798                         if (i < segments.Count - 1)
00799                         {
00800                             newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
      newCmap4.StartCode[i]);
00801 
00802                             for (int j = newCmap4.StartCode[i]; j <= newCmap4.EndCode[i]; j++)
00803                             {
00804                                 includedGlyphs.Add((char)j);
00805                             }
00806                         }
00807                         else
00808                         {
00809                             newCmap4.IdDelta[i] = 1;
00810                             newCmap4.IdRangeOffset[i] = 0;
00811                         }
00812                     }
00813                 }
00814                 else
00815                 {
00816                     for (int i = 0; i < segments.Count - 1; i++)
00817                     {
00818                         newCmap4.EndCode[i] = (ushort)(32 + includedGlyphs.Count + segments[i].Item2 -
      1);
00819                         newCmap4.StartCode[i] = (ushort)(32 + includedGlyphs.Count);
00820                         newCmap4.IdRangeOffset[i] = 0;
00821 
00822                         if (i < segments.Count - 1)
00823                         {
00824                             newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
      newCmap4.StartCode[i]);
00825 
00826                             for (int j = segments[i].Item1; j < segments[i].Item1 + segments[i].Item2;
```

```
j++)
00827                               {
00828                                   includedGlyphs.Add((char)j);
00829                               }
00830                           }
00831                           else
00832                           {
00833                               newCmap4.IdDelta[i] = 1;
00834                               newCmap4.IdRangeOffset[i] = 0;
00835                           }
00836                       }
00837
00838                       {
00839                           int i = segments.Count - 1;
00840                           newCmap4.EndCode[i] = (ushort)(segments[i].Item1 + segments[i].Item2 - 1);
00841                           newCmap4.StartCode[i] = (ushort)segments[i].Item1;
00842                           newCmap4.IdRangeOffset[i] = 0;
00843
00844                           if (i < segments.Count - 1)
00845                           {
00846                               newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
    newCmap4.StartCode[i]);
00847
00848                               for (int j = newCmap4.StartCode[i]; j <= newCmap4.EndCode[i]; j++)
00849                               {
00850                                   includedGlyphs.Add((char)j);
00851                               }
00852                           }
00853                           else
00854                           {
00855                               newCmap4.IdDelta[i] = 1;
00856                               newCmap4.IdRangeOffset[i] = 0;
00857                           }
00858                       }
00859
00860                       if (outputEncoding != null)
00861                       {
00862                           for (int i = 1; i < includedGlyphs.Count; i++)
00863                           {
00864                               outputEncoding.Add(includedGlyphs[i], (char)(32 + i));
00865                           }
00866                       }
00867                   }
00868
00869
00870                   newCmap4.GlyphIndexArray = glyphIndexArray.ToArray();
00871                   newCmap4.Length += (ushort)(newCmap4.GlyphIndexArray.Length * 2);
00872
00873                   TrueTypeCmapTable cmap = new TrueTypeCmapTable() { ActualCmapTables = new ICmapTable[]
    { newCmap4 }, NumberSubTables = 1, Version = 0, SubTables = new CmapSubTable[] { new
    CmapSubTable((ushort)platformId, (ushort)platformSpecificID, 12) } };
00874
00875
00876                   List<Glyph> glyphs = new List<Glyph>();
00877                   List<int> originalGlyphIndices = new List<int>();
00878
00879                   for (int i = 0; i < includedGlyphs.Count; i++)
00880                   {
00881                       int index = this.GetGlyphIndex(includedGlyphs[i]);
00882                       originalGlyphIndices.Add(index);
00883                       glyphs.Add(originalGlyf.Glyphs[index].Clone());
00884                   }
00885
00886                   for (int i = 0; i < glyphs.Count; i++)
00887                   {
00888                       if (glyphs[i] is CompositeGlyph)
00889                       {
00890                           for (int j = 0; j < ((CompositeGlyph)glyphs[i]).GlyphIndex.Length; j++)
00891                           {
00892                               if
    (originalGlyphIndices.Contains(((CompositeGlyph)glyphs[i]).GlyphIndex[j]))
00893                               {
00894                                   ((CompositeGlyph)glyphs[i]).GlyphIndex[j] =
    (ushort)originalGlyphIndices.IndexOf(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00895                               }
00896                               else
00897                               {
00898                                   originalGlyphIndices.Add(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00899
    glyphs.Add(originalGlyf.Glyphs[((CompositeGlyph)glyphs[i]).GlyphIndex[j]]);
00900                                   ((CompositeGlyph)glyphs[i]).GlyphIndex[j] =
    (ushort)originalGlyphIndices.IndexOf(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00901                               }
00902                           }
00903                       }
00904                   }
00905
```

```
00906                 TrueTypeGlyfTable glyf = new TrueTypeGlyfTable() { Glyphs = glyphs.ToArray() };
00907
00908                 TrueTypeLocaTable loca = new TrueTypeLocaTable(glyphs.Count, head.IndexToLocFormat ==
      0);
00909
00910                 for (int i = 0; i < glyphs.Count + 1; i++)
00911                 {
00912                     if (i == 0)
00913                     {
00914                         loca.SetOffset(0, 0);
00915                         loca.Lengths[i] = (uint)(glyphs[i].GetBytes()).Length;
00916                     }
00917                     else
00918                     {
00919                         if (i < glyphs.Count)
00920                         {
00921                             loca.Lengths[i] = (uint)(glyphs[i].GetBytes()).Length;
00922                         }
00923
00924                         loca.SetOffset(i, loca.GetOffset(i - 1) + loca.Lengths[i - 1]);
00925                     }
00926                 }
00927
00928                 TrueTypeHHeaTable hhea = new TrueTypeHHeaTable()
00929                 {
00930                     Version = originalHhea.Version,
00931                     Ascent = originalHhea.Ascent,
00932                     Descent = originalHhea.Descent,
00933                     LineGap = originalHhea.LineGap,
00934                     AdvanceWidthMax = originalHhea.AdvanceWidthMax,
00935                     MinLeftSideBearing = originalHhea.MinLeftSideBearing,
00936                     MinRightSideBearing = originalHhea.MinRightSideBearing,
00937                     XMaxExtent = originalHhea.XMaxExtent,
00938                     CaretSlopeRise = originalHhea.CaretSlopeRise,
00939                     CaretSlopeRun = originalHhea.CaretSlopeRun,
00940                     CaretOffset = originalHhea.CaretOffset,
00941                     MetricDataFormat = originalHhea.MetricDataFormat,
00942                     NumOfLongHorMetrics = (ushort)glyphs.Count
00943                 };
00944
00945
00946                 LongHorFixed[] metrics = new LongHorFixed[glyphs.Count];
00947
00948                 for (int i = 0; i < glyphs.Count; i++)
00949                 {
00950                     metrics[i] = this.GetGlyphMetrics(originalGlyphIndices[i]);
00951                 }
00952
00953                 TrueTypeHmtxTable hmtx = new TrueTypeHmtxTable()
00954                 {
00955                     LeftSideBearing = new short[0],
00956                     HMetrics = metrics
00957                 };
00958
00959                 TrueTypeMaxpTable maxp = new TrueTypeMaxpTable()
00960                 {
00961                     Version = originalMaxp.Version,
00962                     NumGlyphs = (ushort)glyphs.Count,
00963                     MaxPoints = originalMaxp.MaxPoints,
00964                     MaxContours = originalMaxp.MaxContours,
00965                     MaxComponentPoints = originalMaxp.MaxComponentPoints,
00966                     MaxComponentContours = originalMaxp.MaxComponentContours,
00967                     MaxZones = originalMaxp.MaxZones,
00968                     MaxTwilightPoints = originalMaxp.MaxTwilightPoints,
00969                     MaxStorage = originalMaxp.MaxStorage,
00970                     MaxFunctionDefs = originalMaxp.MaxFunctionDefs,
00971                     MaxInstructionDefs = originalMaxp.MaxInstructionDefs,
00972                     MaxStackElements = originalMaxp.MaxStackElements,
00973                     MaxSizeOfInstructions = originalMaxp.MaxSizeOfInstructions,
00974                     MaxComponentElements = originalMaxp.MaxComponentElements,
00975                     MaxComponentDepth = originalMaxp.MaxComponentDepth
00976                 };
00977
00978                 Dictionary<string, ITrueTypeTable> newTables = new Dictionary<string,
      ITrueTypeTable>() {
00979                     {"head", head },
00980                     {"hhea", hhea },
00981                     {"maxp", maxp },
00982                 };
00983
00984                 if (Tables.ContainsKey("OS/2"))
00985                 {
00986                     TrueTypeOS2Table originalOS2 = (TrueTypeOS2Table)Tables["OS/2"];
00987
00988                     TrueTypeOS2Table os2 = new TrueTypeOS2Table()
00989                     {
00990                         AchVendID = originalOS2.AchVendID,
```

```
00991                           FsSelection = originalOS2.FsSelection,
00992                           FsType = originalOS2.FsType,
00993                           Panose = originalOS2.Panose,
00994                           SCapHeight = originalOS2.SCapHeight,
00995                           SFamilyClass = originalOS2.SFamilyClass,
00996                           SFamilySubClass = originalOS2.SFamilySubClass,
00997                           STypoAscender = originalOS2.STypoAscender,
00998                           STypoDescender = originalOS2.STypoDescender,
00999                           STypoLineGap = originalOS2.STypoLineGap,
01000                           SxHeight = originalOS2.SxHeight,
01001                           UlCodePageRange = originalOS2.UlCodePageRange,
01002                           UlUnicodeRange = originalOS2.UlUnicodeRange,
01003                           UsLowerOpticalPointSize = originalOS2.UsLowerOpticalPointSize,
01004                           UsBreakChar = originalOS2.UsBreakChar,
01005                           UsDefaultChar = originalOS2.UsDefaultChar,
01006                           UsMaxContext = originalOS2.UsMaxContext,
01007                           UsUpperOpticalPointSize = originalOS2.UsUpperOpticalPointSize,
01008                           UsWeightClass = originalOS2.UsWeightClass,
01009                           UsWidthClass = originalOS2.UsWidthClass,
01010                           UsWinAscent = originalOS2.UsWinAscent,
01011                           UsWinDescent = originalOS2.UsWinDescent,
01012                           Version = originalOS2.Version,
01013                           XAvgCharWidth = originalOS2.XAvgCharWidth,
01014                           YStrikeoutPosition = originalOS2.YStrikeoutPosition,
01015                           YStrikeoutSize = originalOS2.YStrikeoutSize,
01016                           YSubscriptXOffset = originalOS2.YSubscriptXOffset,
01017                           YSubscriptXSize = originalOS2.YSubscriptXSize,
01018                           YSubscriptYOffset = originalOS2.YSubscriptYOffset,
01019                           YSubscriptYSize = originalOS2.YSubscriptYSize,
01020                           YSuperscriptXOffset = originalOS2.YSuperscriptXOffset,
01021                           YSuperscriptXSize = originalOS2.YSuperscriptXSize,
01022                           YSuperscriptYOffset = originalOS2.YSuperscriptYOffset,
01023                           YSuperscriptYSize = originalOS2.YSuperscriptYSize,
01024                           FsFirstCharIndex = (ushort)characterCodes[0],
01025                           FsLastCharIndex = (ushort)characterCodes[characterCodes.Count - 1]
01026                       };
01027
01028                       newTables.Add("OS/2", os2);
01029                   }
01030
01031                   newTables.Add("hmtx", hmtx);
01032                   newTables.Add("cmap", cmap);
01033                   if (Tables.ContainsKey("fpgm"))
01034                   {
01035                       TrueTypeRawTable fpgm = (TrueTypeRawTable)Tables["fpgm"];
01036                       newTables.Add("fpgm", fpgm);
01037                   }
01038                   if (Tables.ContainsKey("prep"))
01039                   {
01040                       TrueTypeRawTable prep = (TrueTypeRawTable)Tables["prep"];
01041                       newTables.Add("prep", prep);
01042                   }
01043                   if (Tables.ContainsKey("cvt "))
01044                   {
01045                       TrueTypeRawTable cvt = (TrueTypeRawTable)Tables["cvt "];
01046                       newTables.Add("cvt ", cvt);
01047                   }
01048                   newTables.Add("loca", loca);
01049                   newTables.Add("glyf", glyf);
01050
01051                   if (Tables.ContainsKey("name"))
01052                   {
01053                       TrueTypeNameTable name = (TrueTypeNameTable)Tables["name"];
01054                       newTables.Add("name", name);
01055                   }
01056
01057                   if (Tables.ContainsKey("post"))
01058                   {
01059                       TrueTypePostTable oldPost = (TrueTypePostTable)Tables["post"];
01060
01061                       TrueTypePostTable post = new TrueTypePostTable()
01062                       {
01063                           //We don't really care about PostScript names
01064                           Version = new Fixed(0x00030000, 16),
01065                           ItalicAngle = oldPost.ItalicAngle,
01066                           UnderlinePosition = oldPost.UnderlinePosition,
01067                           UnderlineThickness = oldPost.UnderlineThickness,
01068                           IsFixedPitch = oldPost.IsFixedPitch,
01069                           MinMemType42 = oldPost.MinMemType42,
01070                           MaxMemType42 = oldPost.MaxMemType42,
01071                           MinMemType1 = oldPost.MinMemType1,
01072                           MaxMemType1 = oldPost.MaxMemType1
01073                       };
01074
01075                       newTables.Add("post", post);
01076                   }
01077
```

```
01078                 TrueTypeFile newFile = new TrueTypeFile(newTables);
01079
01080                 return newFile;
01081             }
01082
01083         }
01084
01085         internal class TrueTypeGlyfTable :  ITrueTypeTable
01086         {
01087             public Glyph[] Glyphs;
01088
01089             public byte[] GetBytes()
01090             {
01091                 using (MemoryStream ms = new MemoryStream())
01092                 {
01093                     for (int i = 0; i < Glyphs.Length; i++)
01094                     {
01095                         byte[] glyph = Glyphs[i].GetBytes();
01096                         ms.Write(glyph, 0, glyph.Length);
01097                     }
01098                     return ms.ToArray();
01099                 }
01100             }
01101
01102         }
01103
01104         internal abstract class Glyph
01105         {
01106             public short NumberOfContours { get; set; }
01107             public short XMin { get; set; }
01108             public short YMin { get; set; }
01109             public short XMax { get; set; }
01110             public short YMax { get; set; }
01111
01112             public abstract byte[] GetBytes();
01113             public abstract Glyph Clone();
01114             public static Glyph Parse(Stream sr)
01115             {
01116                 short numOfContours = sr.ReadShort();
01117                 if (numOfContours >= 0)
01118                 {
01119                     return new SimpleGlyph(sr, numOfContours);
01120                 }
01121                 else
01122                 {
01123                     return new CompositeGlyph(sr, numOfContours);
01124                 }
01125             }
01126
01127             public abstract TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection);
01128         }
01129
01130         internal class EmptyGlyph :  Glyph
01131         {
01132             public override byte[] GetBytes()
01133             {
01134                 return new byte[0];
01135             }
01136
01137             public override Glyph Clone()
01138             {
01139                 return new EmptyGlyph();
01140             }
01141
01142             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection)
01143             {
01144                 return new TrueTypePoint[0][];
01145             }
01146         }
01147         internal class CompositeGlyph :  Glyph
01148         {
01149             public ushort[] Flags { get; set; }
01150             public ushort[] GlyphIndex { get; set; }
01151             public byte[][] Argument1 { get; set; }
01152             public byte[][] Argument2 { get; set; }
01153             public byte[][] TransformationOption { get; set; }
01154             public ushort NumInstructions { get; set; }
01155             public byte[] Instructions { get; set; }
01156
01157             private CompositeGlyph() { }
01158             public override Glyph Clone()
01159             {
01160                 CompositeGlyph tbr = new CompositeGlyph()
01161                 {
01162                     NumberOfContours = this.NumberOfContours,
```

```
01163                       XMin = this.XMin,
01164                       YMin = this.YMin,
01165                       XMax = this.XMax,
01166                       YMax = this.YMax,
01167 
01168                       Flags = new ushort[this.Flags.Length],
01169 
01170                       GlyphIndex = new ushort[this.GlyphIndex.Length],
01171                       Argument1 = new byte[this.Argument1.Length][],
01172                       Argument2 = new byte[this.Argument2.Length][],
01173                       TransformationOption = new byte[this.TransformationOption.Length][],
01174                       NumInstructions = this.NumInstructions
01175                   };
01176                   this.Flags.CopyTo(tbr.Flags, 0);
01177 
01178                   this.GlyphIndex.CopyTo(tbr.GlyphIndex, 0);
01179 
01180                   for (int i = 0; i < this.Argument1.Length; i++)
01181                   {
01182                       tbr.Argument1[i] = (byte[])this.Argument1[i].Clone();
01183                   }
01184 
01185 
01186                   for (int i = 0; i < this.Argument2.Length; i++)
01187                   {
01188                       tbr.Argument2[i] = (byte[])this.Argument2[i].Clone();
01189                   }
01190 
01191 
01192                   for (int i = 0; i < this.TransformationOption.Length; i++)
01193                   {
01194                       tbr.TransformationOption[i] = (byte[])this.TransformationOption[i].Clone();
01195                   }
01196 
01197 
01198 
01199                   if (this.Instructions != null)
01200                   {
01201                       tbr.Instructions = new byte[this.Instructions.Length];
01202                       this.Instructions.CopyTo(tbr.Instructions, 0);
01203                   }
01204                   else
01205                   {
01206                       tbr.Instructions = null;
01207                   }
01208 
01209                   return tbr;
01210               }
01211               public CompositeGlyph(Stream sr, short numberOfContours) :  base()
01212               {
01213                   this.NumberOfContours = numberOfContours;
01214                   this.XMin = sr.ReadShort();
01215                   this.YMin = sr.ReadShort();
01216                   this.XMax = sr.ReadShort();
01217                   this.YMax = sr.ReadShort();
01218 
01219                   List<ushort> flags = new List<ushort>();
01220                   List<ushort> glyphIndex = new List<ushort>();
01221                   List<byte[]> argument1 = new List<byte[]>();
01222                   List<byte[]> argument2 = new List<byte[]>();
01223                   List<byte[]> transformationOption = new List<byte[]>();
01224 
01225                   bool moreComponents = true;
01226 
01227                   while (moreComponents)
01228                   {
01229                       flags.Add(sr.ReadUShort());
01230 
01231                       moreComponents = (flags.Last() & 0x0020) != 0;
01232                       glyphIndex.Add(sr.ReadUShort());
01233 
01234                       if ((flags.Last() & 0x0001) != 0)
01235                       {
01236                           argument1.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01237                           argument2.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01238                       }
01239                       else
01240                       {
01241                           argument1.Add(new byte[] { (byte)sr.ReadByte() });
01242                           argument2.Add(new byte[] { (byte)sr.ReadByte() });
01243                       }
01244 
01245                       if ((flags.Last() & 0x0008) != 0)
01246                       {
01247                           transformationOption.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte()
      });
01248                       }
```

```
01249                     else if ((flags.Last() & 0x0040) != 0)
01250                     {
01251                         transformationOption.Add(new byte[] { (byte)sr.ReadByte(),
      (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01252                     }
01253                     else if ((flags.Last() & 0x0080) != 0)
01254                     {
01255                         transformationOption.Add(new byte[] { (byte)sr.ReadByte(),
      (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte(),
      (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01256                     }
01257                     else
01258                     {
01259                         transformationOption.Add(new byte[] { });
01260                     }
01261                 }
01262
01263                 this.Flags = flags.ToArray();
01264                 this.GlyphIndex = glyphIndex.ToArray();
01265                 this.Argument1 = argument1.ToArray();
01266                 this.Argument2 = argument2.ToArray();
01267                 this.TransformationOption = transformationOption.ToArray();
01268
01269                 if ((flags.Last() & 0x0100) != 0)
01270                 {
01271                     this.NumInstructions = sr.ReadUShort();
01272                     this.Instructions = new byte[this.NumInstructions];
01273                     sr.Read(this.Instructions, 0, this.NumInstructions);
01274                 }
01275                 else
01276                 {
01277                     this.NumInstructions = 0;
01278                     this.Instructions = null;
01279                 }
01280             }
01281
01282             public override byte[] GetBytes()
01283             {
01284                 using (MemoryStream ms = new MemoryStream())
01285                 {
01286                     ms.WriteShort(this.NumberOfContours);
01287                     ms.WriteShort(this.XMin);
01288                     ms.WriteShort(this.YMin);
01289                     ms.WriteShort(this.XMax);
01290                     ms.WriteShort(this.YMax);
01291
01292                     for (int i = 0; i < this.Flags.Length; i++)
01293                     {
01294                         ms.WriteUShort(this.Flags[i]);
01295                         ms.WriteUShort(this.GlyphIndex[i]);
01296                         ms.Write(this.Argument1[i], 0, this.Argument1[i].Length);
01297                         ms.Write(this.Argument2[i], 0, this.Argument2[i].Length);
01298                         ms.Write(this.TransformationOption[i], 0,
      this.TransformationOption[i].Length);
01299                     }
01300
01301                     if (this.NumInstructions > 0)
01302                     {
01303                         ms.WriteUShort(this.NumInstructions);
01304                         ms.Write(this.Instructions, 0, this.NumInstructions);
01305                     }
01306
01307                     if (ms.Length % 2 != 0)
01308                     {
01309                         ms.WriteByte(0);
01310                     }
01311
01312                     return ms.ToArray();
01313                 }
01314             }
01315
01316             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection)
01317             {
01318                 List<short> argument1 = new List<short>();
01319                 List<short> argument2 = new List<short>();
01320
01321                 for (int i = 0; i < this.Argument1.Length; i++)
01322                 {
01323                     if (this.Argument1[i].Length == 1)
01324                     {
01325                         argument1.Add(this.Argument1[i][0]);
01326                     }
01327                     else if (this.Argument1[i].Length == 2)
01328                     {
01329                         argument1.Add((short)((this.Argument1[i][0] « 8) + this.Argument1[i][1]));
01330                     }
```

```
01331                 }
01332
01333                 for (int i = 0; i < this.Argument2.Length; i++)
01334                 {
01335                     if (this.Argument2[i].Length == 1)
01336                     {
01337                         argument2.Add(this.Argument2[i][0]);
01338                     }
01339                     else if (this.Argument2[i].Length == 2)
01340                     {
01341                         argument2.Add((short)((this.Argument2[i][0] « 8) + this.Argument2[i][1]));
01342                     }
01343                 }
01344
01345                 List<double[]> transformationOption = new List<double[]>();
01346
01347                 for (int i = 0; i < this.TransformationOption.Length; i++)
01348                 {
01349                     if (this.TransformationOption[i].Length == 0)
01350                     {
01351                         transformationOption.Add(new double[] { });
01352                     }
01353                     else if (this.TransformationOption[i].Length == 2)
01354                     {
01355                         double val = (double)((this.TransformationOption[i][0] « 8) +
    this.TransformationOption[i][1]) / (1 « 14);
01356                         transformationOption.Add(new double[] { val });
01357                     }
01358                     else if (this.TransformationOption[i].Length == 4)
01359                     {
01360                         double val1 = (double)((this.TransformationOption[i][0] « 8) +
    this.TransformationOption[i][1]) / (1 « 14);
01361                         double val2 = (double)((this.TransformationOption[i][2] « 8) +
    this.TransformationOption[i][3]) / (1 « 14);
01362                         transformationOption.Add(new double[] { val1, val2 });
01363                     }
01364                     else if (this.TransformationOption[i].Length == 8)
01365                     {
01366                         double val1 = (double)((this.TransformationOption[i][0] « 8) +
    this.TransformationOption[i][1]) / (1 « 14);
01367                         double val2 = (double)((this.TransformationOption[i][2] « 8) +
    this.TransformationOption[i][3]) / (1 « 14);
01368                         double val3 = (double)((this.TransformationOption[i][4] « 8) +
    this.TransformationOption[i][5]) / (1 « 14);
01369                         double val4 = (double)((this.TransformationOption[i][6] « 8) +
    this.TransformationOption[i][7]) / (1 « 14);
01370                         transformationOption.Add(new double[] { val1, val2, val3, val4 });
01371                     }
01372                 }
01373
01374                 List<TrueTypePoint[]> tbr = new List<TrueTypePoint[]>();
01375
01376
01377                 for (int i = 0; i < this.GlyphIndex.Length; i++)
01378                 {
01379                     TrueTypePoint[][] componentContours =
    glyphCollection[this.GlyphIndex[i]].GetGlyphPath(size, emSize, glyphCollection);
01380
01381                     double[,] transformMatrix = new double[,] { { 1, 0 }, { 0, 1 } };
01382
01383                     if ((Flags[i] & 0x0008) != 0)
01384                     {
01385                         transformMatrix[0, 0] = transformationOption[i][0];
01386                         transformMatrix[1, 1] = transformationOption[i][0];
01387                     }
01388                     else if ((Flags[i] & 0x0040) != 0)
01389                     {
01390                         transformMatrix[0, 0] = transformationOption[i][0];
01391                         transformMatrix[1, 1] = transformationOption[i][1];
01392                     }
01393                     else if ((Flags[i] & 0x0080) != 0)
01394                     {
01395                         transformMatrix[0, 0] = transformationOption[i][0];
01396                         transformMatrix[0, 1] = transformationOption[i][1];
01397                         transformMatrix[1, 0] = transformationOption[i][2];
01398                         transformMatrix[1, 1] = transformationOption[i][3];
01399                     }
01400
01401                     double deltaX = 0;
01402                     double deltaY = 0;
01403
01404                     if ((Flags[i] & 0x0002) != 0)
01405                     {
01406                         deltaX = argument1[i] * size / emSize;
01407                         deltaY = argument2[i] * size / emSize;
01408
01409                         if ((Flags[i] & 0x0800) != 0 && (Flags[i] & 0x1000) == 0)
```

```
01410                         {
01411                             deltaX *= Magnitude(Multiply(transformMatrix, new double[] { 1, 0 }));
01412                             deltaY *= Magnitude(Multiply(transformMatrix, new double[] { 0, 1 }));
01413                         }
01414                     }
01415                     else
01416                     {
01417                         TrueTypePoint reference = GetNthElementWhere(tbr, argument1[i], el =>
      el.IsDefinedPoint);
01418                         TrueTypePoint destination = GetNthElementWhere(componentContours,
      argument2[i], el => el.IsDefinedPoint);
01419 
01420                         deltaX = reference.X - destination.X;
01421                         deltaY = reference.Y - destination.Y;
01422 
01423                         //Note:  this would be the sensible behaviour (i.e.  make sure that reference
      and destination coincide after the transform), but apparently it is not the correct one.
01424                         /*double[] transfDestination = Multiply(transformMatrix, new double[] {
      destination.X, destination.Y });
01425 
01426 deltaX = reference.X - transfDestination[0];
01427 deltaY = reference.Y - transfDestination[1];*/
01428                     }
01429 
01430                     for (int j = 0; j < componentContours.Length; j++)
01431                     {
01432                         for (int k = 0; k < componentContours[j].Length; k++)
01433                         {
01434                             double[] transformed = Multiply(transformMatrix, new double[] {
      componentContours[j][k].X, componentContours[j][k].Y });
01435 
01436                             componentContours[j][k] = new TrueTypePoint(transformed[0] + deltaX,
      transformed[1] + deltaY, componentContours[j][k].IsOnCurve, componentContours[j][k].IsDefinedPoint);
01437                         }
01438                     }
01439 
01440                     tbr.AddRange(componentContours);
01441                 }
01442 
01443                 return tbr.ToArray();
01444             }
01445         }
01446 
01447         private static double Magnitude(double[] vector)
01448         {
01449             double tbr = 0;
01450 
01451             for (int i = 0; i < vector.Length; i++)
01452             {
01453                 tbr += vector[i] * vector[i];
01454             }
01455 
01456             return Math.Sqrt(tbr);
01457         }
01458 
01459         private static double[] Multiply(double[,] matrix, double[] vector)
01460         {
01461             double[] tbr = new double[2];
01462 
01463             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1];
01464             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1];
01465 
01466             return tbr;
01467         }
01468 
01469         private static T GetNthElementWhere<T>(IEnumerable<IEnumerable<T» array, int n, Func<T, bool>
      condition)
01470         {
01471             int index = 0;
01472 
01473             foreach (IEnumerable<T> arr1 in array)
01474             {
01475                 foreach (T el in arr1)
01476                 {
01477                     if (condition(el))
01478                     {
01479                         if (index == n)
01480                         {
01481                             return el;
01482                         }
01483                         index++;
01484                     }
01485                 }
01486             }
01487 
01488             throw new IndexOutOfRangeException();
01489         }
```

```
01490
01491 /// <summary>
01492 /// Represents a point in a TrueType path description.
01493 /// </summary>
01494         public struct TrueTypePoint
01495         {
01496 /// <summary>
01497 /// The horizontal coordinate of the point.
01498 /// </summary>
01499             public double X;
01500
01501 /// <summary>
01502 /// The vertical coordinate of the point.
01503 /// </summary>
01504             public double Y;
01505
01506 /// <summary>
01507 /// Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.
01508 /// </summary>
01509             public bool IsOnCurve;
01510
01511             internal bool IsDefinedPoint;
01512
01513             internal TrueTypePoint(double x, double y, bool onCurve, bool isDefinedPoint)
01514             {
01515                 this.X = x;
01516                 this.Y = y;
01517                 this.IsOnCurve = onCurve;
01518                 this.IsDefinedPoint = isDefinedPoint;
01519             }
01520         }
01521
01522         internal class SimpleGlyph :  Glyph
01523         {
01524             public ushort[] EndPtsOfContours { get; set; }
01525             public ushort InstructionLength { get; set; }
01526             public byte[] Instructions { get; set; }
01527             public byte[] Flags { get; set; }
01528             public byte[] XCoordinates { get; set; }
01529             public byte[] YCoordinates { get; set; }
01530
01531             private SimpleGlyph() { }
01532             public override Glyph Clone()
01533             {
01534                 SimpleGlyph tbr = new SimpleGlyph
01535                 {
01536                     NumberOfContours = this.NumberOfContours,
01537                     XMin = this.XMin,
01538                     YMin = this.YMin,
01539                     XMax = this.XMax,
01540                     YMax = this.YMax,
01541
01542                     EndPtsOfContours = new ushort[this.EndPtsOfContours.Length],
01543                     InstructionLength = this.InstructionLength,
01544                     Instructions = new byte[this.Instructions.Length],
01545                     Flags = new byte[this.Flags.Length],
01546                     XCoordinates = new byte[this.XCoordinates.Length],
01547                     YCoordinates = new byte[this.YCoordinates.Length]
01548                 };
01549
01550                 this.EndPtsOfContours.CopyTo(tbr.EndPtsOfContours, 0);
01551                 this.Instructions.CopyTo(tbr.Instructions, 0);
01552                 this.Flags.CopyTo(tbr.Flags, 0);
01553                 this.XCoordinates.CopyTo(tbr.XCoordinates, 0);
01554                 this.YCoordinates.CopyTo(tbr.YCoordinates, 0);
01555
01556                 return tbr;
01557             }
01558
01559             public SimpleGlyph(Stream sr, short numberOfContours) :  base()
01560             {
01561                 this.NumberOfContours = numberOfContours;
01562                 this.XMin = sr.ReadShort();
01563                 this.YMin = sr.ReadShort();
01564                 this.XMax = sr.ReadShort();
01565                 this.YMax = sr.ReadShort();
01566
01567                 this.EndPtsOfContours = new ushort[this.NumberOfContours];
01568                 for (int i = 0; i < this.NumberOfContours; i++)
01569                 {
01570                     this.EndPtsOfContours[i] = sr.ReadUShort();
01571                 }
01572
01573                 this.InstructionLength = sr.ReadUShort();
01574                 this.Instructions = new byte[this.InstructionLength];
01575                 for (int i = 0; i < this.InstructionLength; i++)
01576                 {
```

```
01577                   this.Instructions[i] = (byte)sr.ReadByte();
01578               }
01579
01580               List<byte> logicalFlags = new List<byte>();
01581               List<byte> flags = new List<byte>();
01582
01583               int totalPoints = this.EndPtsOfContours[this.NumberOfContours - 1] + 1;
01584
01585               int countedPoints = 0;
01586
01587               while (countedPoints < totalPoints)
01588               {
01589                   flags.Add((byte)sr.ReadByte());
01590                   logicalFlags.Add(flags.Last());
01591                   countedPoints++;
01592                   if ((flags.Last() & 0x08) != 0)
01593                   {
01594                       byte repeats = (byte)sr.ReadByte();
01595                       for (int i = 0; i < repeats; i++)
01596                       {
01597                           logicalFlags.Add(flags.Last());
01598                           countedPoints++;
01599                       }
01600                       flags.Add(repeats);
01601                   }
01602               }
01603
01604               this.Flags = flags.ToArray();
01605
01606               List<byte> xCoordinates = new List<byte>();
01607
01608               for (int i = 0; i < totalPoints; i++)
01609               {
01610                   bool isByte = (logicalFlags[i] & 0x02) != 0;
01611
01612                   if (isByte)
01613                   {
01614                       xCoordinates.Add((byte)sr.ReadByte());
01615                   }
01616                   else if ((logicalFlags[i] & 0x10) == 0)
01617                   {
01618                       xCoordinates.Add((byte)sr.ReadByte());
01619                       xCoordinates.Add((byte)sr.ReadByte());
01620                   }
01621               }
01622
01623               this.XCoordinates = xCoordinates.ToArray();
01624
01625               List<byte> yCoordinates = new List<byte>();
01626
01627               List<int> yCoordinateLengths = new List<int>();
01628
01629               for (int i = 0; i < totalPoints; i++)
01630               {
01631                   bool isByte = (logicalFlags[i] & 0x04) != 0;
01632
01633                   if (isByte)
01634                   {
01635                       yCoordinates.Add((byte)sr.ReadByte());
01636                       yCoordinateLengths.Add(1);
01637                   }
01638                   else if ((logicalFlags[i] & 0x20) == 0)
01639                   {
01640                       yCoordinates.Add((byte)sr.ReadByte());
01641                       yCoordinates.Add((byte)sr.ReadByte());
01642                       yCoordinateLengths.Add(2);
01643                   }
01644                   else
01645                   {
01646                       yCoordinateLengths.Add(0);
01647                   }
01648               }
01649
01650               this.YCoordinates = yCoordinates.ToArray();
01651           }
01652
01653           public override byte[] GetBytes()
01654           {
01655               using (MemoryStream ms = new MemoryStream())
01656               {
01657                   ms.WriteShort(this.NumberOfContours);
01658                   ms.WriteShort(this.XMin);
01659                   ms.WriteShort(this.YMin);
01660                   ms.WriteShort(this.XMax);
01661                   ms.WriteShort(this.YMax);
01662
01663                   for (int i = 0; i < this.EndPtsOfContours.Length; i++)
```

```
01664                     {
01665                         ms.WriteUShort(this.EndPtsOfContours[i]);
01666                     }
01667 
01668                     ms.WriteUShort(this.InstructionLength);
01669                     ms.Write(this.Instructions, 0, this.InstructionLength);
01670                     ms.Write(this.Flags, 0, this.Flags.Length);
01671                     ms.Write(this.XCoordinates, 0, this.XCoordinates.Length);
01672                     ms.Write(this.YCoordinates, 0, this.YCoordinates.Length);
01673 
01674                     if (ms.Length % 2 != 0)
01675                     {
01676                         ms.WriteByte(0);
01677                     }
01678 
01679                     return ms.ToArray();
01680                 }
01681             }
01682 
01683             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection)
01684             {
01685                 List<TrueTypePoint[]> contours = new List<TrueTypePoint[]>();
01686 
01687                 List<TrueTypePoint> currentContour = new List<TrueTypePoint>();
01688 
01689 
01690                 List<byte> logicalFlags = new List<byte>();
01691 
01692                 int totalPoints = this.EndPtsOfContours[this.NumberOfContours - 1] + 1;
01693 
01694                 int countedPoints = 0;
01695 
01696                 int index = 0;
01697 
01698                 while (countedPoints < totalPoints)
01699                 {
01700                     logicalFlags.Add(this.Flags[index]);
01701                     index++;
01702                     countedPoints++;
01703                     if ((logicalFlags.Last() & 0x08) != 0)
01704                     {
01705                         byte repeats = this.Flags[index];
01706                         index++;
01707                         for (int i = 0; i < repeats; i++)
01708                         {
01709                             logicalFlags.Add(logicalFlags.Last());
01710                             countedPoints++;
01711                         }
01712                     }
01713                 }
01714 
01715                 List<short> xCoordinates = new List<short>();
01716 
01717                 index = 0;
01718 
01719                 for (int i = 0; i < totalPoints; i++)
01720                 {
01721                     bool isByte = (logicalFlags[i] & 0x02) != 0;
01722 
01723                     if (isByte)
01724                     {
01725                         if ((logicalFlags[i] & 0x10) != 0)
01726                         {
01727                             xCoordinates.Add(this.XCoordinates[index]);
01728                         }
01729                         else
01730                         {
01731                             xCoordinates.Add((short)(-this.XCoordinates[index]));
01732                         }
01733 
01734                         index++;
01735                     }
01736                     else if ((logicalFlags[i] & 0x10) == 0)
01737                     {
01738                         xCoordinates.Add((short)((this.XCoordinates[index] << 8) +
      this.XCoordinates[index + 1]));
01739                         index += 2;
01740                     }
01741                     else
01742                     {
01743                         xCoordinates.Add(0);
01744                     }
01745                 }
01746 
01747                 List<short> yCoordinates = new List<short>();
01748 
```

```
01749                 index = 0;
01750
01751                 for (int i = 0; i < totalPoints; i++)
01752                 {
01753                     bool isByte = (logicalFlags[i] & 0x04) != 0;
01754
01755                     if (isByte)
01756                     {
01757                         if ((logicalFlags[i] & 0x20) != 0)
01758                         {
01759                             yCoordinates.Add(this.YCoordinates[index]);
01760                         }
01761                         else
01762                         {
01763                             yCoordinates.Add((short)(-this.YCoordinates[index]));
01764                         }
01765                         index++;
01766                     }
01767                     else if ((logicalFlags[i] & 0x20) == 0)
01768                     {
01769                         yCoordinates.Add((short)((this.YCoordinates[index] << 8) +
      this.YCoordinates[index + 1]));
01770                         index += 2;
01771                     }
01772                     else
01773                     {
01774                         yCoordinates.Add(0);
01775                     }
01776                 }
01777
01778                 int[] previousPoint = new int[2] { 0, 0 };
01779
01780                 for (int i = 0; i < totalPoints; i++)
01781                 {
01782                     int absoluteX = xCoordinates[i] + previousPoint[0];
01783                     int absoluteY = yCoordinates[i] + previousPoint[1];
01784
01785                     previousPoint[0] = absoluteX;
01786                     previousPoint[1] = absoluteY;
01787
01788                     bool onCurve = (logicalFlags[i] & 0x01) != 0;
01789
01790                     if (onCurve)
01791                     {
01792                         currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01793                     }
01794                     else
01795                     {
01796                         if (currentContour.Count > 0)
01797                         {
01798                             if (currentContour.Last().IsOnCurve)
01799                             {
01800                                 currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01801                             }
01802                             else
01803                             {
01804                                 double newX = size * absoluteX / emSize;
01805                                 double newY = size * absoluteY / emSize;
01806
01807                                 currentContour.Add(new TrueTypePoint((newX + currentContour.Last().X)
      * 0.5, (newY + currentContour.Last().Y) * 0.5, true, false));
01808                                 currentContour.Add(new TrueTypePoint(newX, newY, onCurve, true));
01809                             }
01810                         }
01811                         else
01812                         {
01813                             currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01814                         }
01815                     }
01816
01817                     if (this.EndPtsOfContours.Contains((ushort)i))
01818                     {
01819                         if (!currentContour[0].IsOnCurve)
01820                         {
01821                             if (currentContour.Last().IsOnCurve)
01822                             {
01823                                 currentContour.Insert(0, new TrueTypePoint(currentContour.Last().X,
      currentContour.Last().Y, currentContour.Last().IsOnCurve, false));
01824                             }
01825                             else
01826                             {
01827                                 currentContour.Insert(0, new TrueTypePoint((currentContour[0].X +
      currentContour.Last().X) * 0.5, (currentContour[0].Y + currentContour.Last().Y) * 0.5, true, false));
01828                             }
```

```
01829                         }
01830
01831                         if (!currentContour.Last().IsOnCurve)
01832                         {
01833                             currentContour.Add(new TrueTypePoint(currentContour[0].X,
      currentContour[0].Y, currentContour[0].IsOnCurve, false));
01834                         }
01835
01836                         contours.Add(currentContour.ToArray());
01837                         currentContour = new List<TrueTypePoint>();
01838                     }
01839                 }
01840
01841                 return contours.ToArray();
01842             }
01843         }
01844
01845         internal class TrueTypeLocaTable :  ITrueTypeTable
01846         {
01847             public ushort[] ShortOffsets { get; }
01848             public uint[] IntOffsets { get; }
01849             public uint[] Lengths { get; }
01850
01851             public uint GetOffset(int index)
01852             {
01853                 if (IntOffsets == null)
01854                 {
01855                     return (uint)ShortOffsets[index] * 2;
01856                 }
01857                 else
01858                 {
01859                     return IntOffsets[index];
01860                 }
01861             }
01862
01863             public void SetOffset(int index, uint value)
01864             {
01865                 if (IntOffsets == null)
01866                 {
01867                     ShortOffsets[index] = (ushort)(value / 2);
01868                 }
01869                 else
01870                 {
01871                     IntOffsets[index] = value;
01872                 }
01873             }
01874
01875             public TrueTypeLocaTable(int numGlyphs, bool isShort)
01876             {
01877                 this.Lengths = new uint[numGlyphs];
01878
01879                 if (isShort)
01880                 {
01881                     this.ShortOffsets = new ushort[numGlyphs + 1];
01882                 }
01883                 else
01884                 {
01885                     this.IntOffsets = new uint[numGlyphs + 1];
01886                 }
01887             }
01888
01889             public TrueTypeLocaTable(Stream sr, int numGlyphs, bool isShort)
01890             {
01891                 this.Lengths = new uint[numGlyphs];
01892
01893                 if (isShort)
01894                 {
01895                     this.ShortOffsets = new ushort[numGlyphs + 1];
01896                     for (int i = 0; i < numGlyphs + 1; i++)
01897                     {
01898                         this.ShortOffsets[i] = sr.ReadUShort();
01899                     }
01900
01901                     for (int i = 0; i < numGlyphs; i++)
01902                     {
01903                         this.Lengths[i] = 2 * ((uint)this.ShortOffsets[i + 1] -
      (uint)this.ShortOffsets[i]);
01904                     }
01905                 }
01906                 else
01907                 {
01908                     this.IntOffsets = new uint[numGlyphs + 1];
01909                     for (int i = 0; i < numGlyphs + 1; i++)
01910                     {
01911                         this.IntOffsets[i] = sr.ReadUInt();
01912                     }
01913
```

```
01914                     for (int i = 0; i < numGlyphs; i++)
01915                     {
01916                         this.Lengths[i] = this.IntOffsets[i + 1] - this.IntOffsets[i];
01917                     }
01918                 }
01919 
01920             }
01921 
01922             public byte[] GetBytes()
01923             {
01924                 using (MemoryStream ms = new MemoryStream())
01925                 {
01926 
01927                     if (IntOffsets == null)
01928                     {
01929                         for (int i = 0; i < ShortOffsets.Length; i++)
01930                         {
01931                             ms.WriteUShort(ShortOffsets[i]);
01932                         }
01933                     }
01934                     else
01935                     {
01936                         for (int i = 0; i < IntOffsets.Length; i++)
01937                         {
01938                             ms.WriteUInt(IntOffsets[i]);
01939                         }
01940                     }
01941 
01942                     return ms.ToArray();
01943                 }
01944             }
01945         }
01946 
01947         internal class TrueTypeRawTable :  ITrueTypeTable
01948         {
01949             public byte[] Data { get; }
01950 
01951             public TrueTypeRawTable(Stream sr, uint length)
01952             {
01953                 this.Data = new byte[length];
01954                 sr.Read(this.Data, 0, (int)length);
01955             }
01956 
01957             public byte[] GetBytes()
01958             {
01959                 return Data;
01960             }
01961         }
01962 
01963         internal bool HasCmap4Table()
01964         {
01965             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
01966             {
01967                 if (cmap is CmapTable4)
01968                 {
01969                     return true;
01970                 }
01971 
01972             }
01973             return false;
01974         }
01975 
01976 /// <summary>
01977 /// Obtains the font family name from the TrueType file.
01978 /// </summary>
01979 /// <returns>The font family name, if available; <see langword="null"/> otherwise.</returns>
01980         public string GetFontFamilyName()
01981         {
01982             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
01983 
01984             for (int i = 0; i < name.Count; i++)
01985             {
01986                 if (name.NameRecords[i].NameID == 16)
01987                 {
01988                     return name.Name[i];
01989                 }
01990             }
01991 
01992             for (int i = 0; i < name.Count; i++)
01993             {
01994                 if (name.NameRecords[i].NameID == 1)
01995                 {
01996                     return name.Name[i];
01997                 }
01998             }
01999 
02000             return null;
```

```
02001         }
02002
02003 /// <summary>
02004 /// Obtains the full font family name from the TrueType file.
02005 /// </summary>
02006 /// <returns>The full font family name, if available; <see langword="null"/> otherwise.</returns>
02007         public string GetFullFontFamilyName()
02008         {
02009             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
02010
02011             for (int i = 0; i < name.Count; i++)
02012             {
02013                 if (name.NameRecords[i].NameID == 4)
02014                 {
02015                     return name.Name[i];
02016                 }
02017             }
02018
02019             for (int i = 0; i < name.Count; i++)
02020             {
02021                 if (name.NameRecords[i].NameID == 6)
02022                 {
02023                     return name.Name[i];
02024                 }
02025             }
02026
02027             return null;
02028         }
02029
02030
02031 /// <summary>
02032 /// Obtains the PostScript font name from the TrueType file.
02033 /// </summary>
02034 /// <returns>The PostScript font name, if available; <see langword="null"/> otherwise.</returns>
02035         public string GetFontName()
02036         {
02037             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
02038
02039             for (int i = 0; i < name.Count; i++)
02040             {
02041                 if (name.NameRecords[i].NameID == 6)
02042                 {
02043                     return name.Name[i];
02044                 }
02045             }
02046
02047             return null;
02048         }
02049
02050 /// <summary>
02051 /// Returns the index of the first character glyph represented by the font.
02052 /// </summary>
02053 /// <returns>The index of the first character glyph represented by the font.</returns>
02054         public ushort GetFirstCharIndex()
02055         {
02056             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02057
02058             return os2.FsFirstCharIndex;
02059         }
02060
02061 /// <summary>
02062 /// Returns the index of the last character glyph represented by the font.
02063 /// </summary>
02064 /// <returns>The index of the last character glyph represented by the font.</returns>
02065         public ushort GetLastCharIndex()
02066         {
02067             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02068
02069             return os2.FsLastCharIndex;
02070         }
02071
02072
02073 /// <summary>
02074 /// Determines whether the typeface is Italic or Oblique or not.
02075 /// </summary>
02076 /// <returns>A <see cref="bool"/> indicating whether the typeface is Italic or Oblique or
      not.</returns>
02077         public bool IsItalic()
02078         {
02079             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02080
02081             return (os2.FsSelection & 1) == 1;
02082         }
02083
02084 /// <summary>
02085 /// Determines whether the typeface is Oblique or not.
02086 /// </summary>
```

```
02087 /// <returns>A <see cref="bool"/> indicating whether the typeface is Oblique or not.</returns>
02088         public bool IsOblique()
02089         {
02090             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02091 
02092             return (os2.FsSelection & 512) != 0;
02093         }
02094 
02095 /// <summary>
02096 /// Determines whether the typeface is Bold or not.
02097 /// </summary>
02098 /// <returns>A <see cref="bool"/> indicating whether the typeface is Bold or not</returns>
02099         public bool IsBold()
02100         {
02101             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02102 
02103             return (os2.FsSelection & 32) != 0;
02104         }
02105 
02106 /// <summary>
02107 /// Determines whether the typeface is fixed-pitch (aka monospaces) or not.
02108 /// </summary>
02109 /// <returns>A <see cref="bool"/> indicating whether the typeface is fixed-pitch (aka monospaces) or
      not.</returns>
02110         public bool IsFixedPitch()
02111         {
02112             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02113 
02114             return os2.Panose.BProportion == 9;
02115         }
02116 
02117 /// <summary>
02118 /// Determines whether the typeface is serifed or not.
02119 /// </summary>
02120 /// <returns>A <see cref="bool"/> indicating whether the typeface is serifed or not.</returns>
02121         public bool IsSerif()
02122         {
02123             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02124 
02125             return os2.SFamilyClass == 1 || os2.SFamilyClass == 2 || os2.SFamilyClass == 3 ||
      os2.SFamilyClass == 4 || os2.SFamilyClass == 5 || os2.SFamilyClass == 7;
02126         }
02127 
02128 /// <summary>
02129 /// Determines whether the typeface is a script typeface or not.
02130 /// </summary>
02131 /// <returns>A <see cref="bool"/> indicating whether the typeface is a script typeface or
      not.</returns>
02132         public bool IsScript()
02133         {
02134             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02135 
02136             return os2.SFamilyClass == 10;
02137         }
02138 
02139 /// <summary>
02140 /// Determines the index of the glyph corresponding to a certain character.
02141 /// </summary>
02142 /// <param name="glyph">The character whose glyph is sought.</param>
02143 /// <returns>The index of the glyph in the TrueType file.</returns>
02144         public int GetGlyphIndex(char glyph)
02145         {
02146             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
02147             {
02148                 if (cmap is CmapTable4)
02149                 {
02150                     return cmap.GetGlyphIndex(glyph);
02151                 }
02152             }
02153 
02154             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
02155             {
02156                 if (cmap is CmapTable0)
02157                 {
02158                     return cmap.GetGlyphIndex(glyph);
02159                 }
02160             }
02161 
02162             return -1;
02163         }
02164 
02165         internal int GetGlyphWidth(int glyphIndex)
02166         {
02167             if (((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length > glyphIndex)
02168             {
02169                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics[glyphIndex].AdvanceWidth;
02170             }
```

```
02171             else
02172             {
02173                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Last().AdvanceWidth;
02174             }
02175         }
02176 
02177         internal LongHorFixed GetGlyphMetrics(int glyphIndex)
02178         {
02179             if (((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length > glyphIndex)
02180             {
02181                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics[glyphIndex];
02182             }
02183             else
02184             {
02185                 return new
      LongHorFixed(((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Last().AdvanceWidth,
      ((TrueTypeHmtxTable)this.Tables["hmtx"]).LeftSideBearing[glyphIndex -
      ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length]);
02186             }
02187         }
02188 
02189 /// <summary>
02190 /// Get the path that describes the shape of a glyph.
02191 /// </summary>
02192 /// <param name="glyphIndex">The index of the glyph whose path is sought.</param>
02193 /// <param name="size">The font size to be used for the font coordinates.</param>
02194 /// <returns>An array of contours, each of which is itself an array of TrueType points.</returns>
02195         public TrueTypePoint[][] GetGlyphPath(int glyphIndex, double size)
02196         {
02197             return ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex].GetGlyphPath(size,
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs);
02198         }
02199 
02200 /// <summary>
02201 /// Get the path that describes the shape of a glyph.
02202 /// </summary>
02203 /// <param name="glyph">The glyph whose path is sought.</param>
02204 /// <param name="size">The font size to be used for the font coordinates.</param>
02205 /// <returns>An array of contours, each of which is itself an array of TrueType points.</returns>
02206         public TrueTypePoint[][] GetGlyphPath(char glyph, double size)
02207         {
02208             return
      ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[GetGlyphIndex(glyph)].GetGlyphPath(size,
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs);
02209         }
02210 
02211 /// <summary>
02212 /// Computes the advance width of a glyph, in thousandths of em unit.
02213 /// </summary>
02214 /// <param name="glyph">The glyph whose advance width is to be computed.</param>
02215 /// <returns>The advance width of the glyph in thousandths of em unit.</returns>
02216         public double Get1000EmGlyphWidth(char glyph)
02217         {
02218             int num = (int)glyph;
02219             if (num < 256)
02220             {
02221                 return GlyphWidthsCache[num];
02222             }
02223             else
02224             {
02225                 return Get1000EmGlyphWidth(GetGlyphIndex(glyph));
02226             }
02227         }
02228 
02229 /// <summary>
02230 /// Computes the advance width of a glyph, in thousandths of em unit.
02231 /// </summary>
02232 /// <param name="glyphIndex">The index of the glyph whose advance width is to be computed.</param>
02233 /// <returns>The advance width of the glyph in thousandths of em unit.</returns>
02234         public double Get1000EmGlyphWidth(int glyphIndex)
02235         {
02236             int w = GetGlyphWidth(glyphIndex);
02237 
02238             return w * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02239         }
02240 
02241 /// <summary>
02242 /// Computes the font's Win ascent, in thousandths of em unit.
02243 /// </summary>
02244 /// <returns>The font's Win ascent in thousandths of em unit.</returns>
02245         public double Get1000EmWinAscent()
02246         {
02247             TrueTypeOS2Table os2 = ((TrueTypeOS2Table)this.Tables["OS/2"]);
02248 
02249             bool useTypoMetrics = (os2.FsSelection & 128) != 0;
02250 
02251             if (!useTypoMetrics)
```

```
02252             {
02253                 return ((TrueTypeOS2Table)this.Tables["OS/2"]).UsWinAscent * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02254             }
02255             else
02256             {
02257                 return ((TrueTypeOS2Table)this.Tables["OS/2"]).STypoAscender * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02258             }
02259         }
02260
02261 /// <summary>
02262 /// Computes the font ascent, in thousandths of em unit.
02263 /// </summary>
02264 /// <returns>The font ascent in thousandths of em unit.</returns>
02265         public double Get1000EmAscent()
02266         {
02267             return ((TrueTypeHHeaTable)this.Tables["hhea"]).Ascent * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02268         }
02269
02270
02271 /// <summary>
02272 /// Computes the font descent, in thousandths of em unit.
02273 /// </summary>
02274 /// <returns>The font descent in thousandths of em unit.</returns>
02275         public double Get1000EmDescent()
02276         {
02277             return ((TrueTypeHHeaTable)this.Tables["hhea"]).Descent * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02278         }
02279
02280 /// <summary>
02281 /// Computes the maximum height over the baseline of the font, in thousandths of em unit.
02282 /// </summary>
02283 /// <returns>The maximum height over the baseline of the font in thousandths of em unit.</returns>
02284         public double Get1000EmYMax()
02285         {
02286             return ((TrueTypeHeadTable)this.Tables["head"]).YMax * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02287         }
02288
02289 /// <summary>
02290 /// Computes the maximum depth below the baseline of the font, in thousandths of em unit.
02291 /// </summary>
02292 /// <returns>The maximum depth below the baseline of the font in thousandths of em unit.</returns>
02293         public double Get1000EmYMin()
02294         {
02295             return ((TrueTypeHeadTable)this.Tables["head"]).YMin * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02296         }
02297
02298 /// <summary>
02299 /// Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em
    unit.
02300 /// </summary>
02301 /// <returns>The maximum distance to the right of the glyph origin of the font in thousandths of em
    unit.</returns>
02302         public double Get1000EmXMax()
02303         {
02304             return ((TrueTypeHeadTable)this.Tables["head"]).XMax * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02305         }
02306
02307 /// <summary>
02308 /// Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em
    unit.
02309 /// </summary>
02310 /// <returns>The maximum distance to the left of the glyph origin of the font in thousandths of em
    unit.</returns>
02311         public double Get1000EmXMin()
02312         {
02313             return ((TrueTypeHeadTable)this.Tables["head"]).XMin * 1000.0 /
    ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02314         }
02315
02316 /// <summary>
02317 /// Represents the left- and right-side bearings of a glyph.
02318 /// </summary>
02319         public struct Bearings
02320         {
02321 /// <summary>
02322 /// The left-side bearing of the glyph.
02323 /// </summary>
02324             public int LeftSideBearing;
02325
02326 /// <summary>
```

```
02327 /// The right-side bearing of the glyph.
02328 /// </summary>
02329             public int RightSideBearing;
02330 
02331             internal Bearings(int lsb, int rsb)
02332             {
02333                 LeftSideBearing = lsb;
02334                 RightSideBearing = rsb;
02335             }
02336         }
02337 
02338         internal Bearings Get1000EmGlyphBearings(int glyphIndex)
02339         {
02340             LongHorFixed metrics = GetGlyphMetrics(glyphIndex);
02341 
02342             int lsb = metrics.LeftSideBearing;
02343 
02344             Glyph glyph = ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex];
02345 
02346             int rsb = metrics.AdvanceWidth - (lsb + glyph.XMax - glyph.XMin);
02347 
02348             return new Bearings(lsb * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, rsb
      * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm);
02349         }
02350 
02351 
02352 /// <summary>
02353 /// Computes the left- and right- side bearings of a glyph, in thousandths of em unit.
02354 /// </summary>
02355 /// <param name="glyph">The glyph whose bearings are to be computed.</param>
02356 /// <returns>The left- and right- side bearings of the glyph in thousandths of em unit</returns>
02357         public Bearings Get1000EmGlyphBearings(char glyph)
02358         {
02359             int num = (int)glyph;
02360 
02361             if (num < 256)
02362             {
02363                 return BearingsCache[num];
02364             }
02365             else
02366             {
02367                 return Get1000EmGlyphBearings(GetGlyphIndex(glyph));
02368             }
02369         }
02370 
02371 /// <summary>
02372 /// Represents the maximum heigth above and depth below the baseline of a glyph.
02373 /// </summary>
02374         public struct VerticalMetrics
02375         {
02376 /// <summary>
02377 /// The maximum depth below the baseline of the glyph.
02378 /// </summary>
02379             public int YMin;
02380 
02381 /// <summary>
02382 /// The maximum height above the baseline of the glyph.
02383 /// </summary>
02384             public int YMax;
02385 
02386             internal VerticalMetrics(int yMin, int yMax)
02387             {
02388                 this.YMin = yMin;
02389                 this.YMax = yMax;
02390             }
02391         }
02392 
02393         internal VerticalMetrics Get1000EmGlyphVerticalMetrics(int glyphIndex)
02394         {
02395             Glyph glyph = ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex];
02396 
02397             return new VerticalMetrics(glyph.YMin * 1000 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, glyph.YMax * 1000 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm);
02398         }
02399 
02400 /// <summary>
02401 /// Computes the vertical metrics of a glyph, in thousandths of em unit.
02402 /// </summary>
02403 /// <param name="glyph">The glyph whose vertical metrics are to be computed.</param>
02404 /// <returns>The vertical metrics of a glyph, in thousandths of em unit.</returns>
02405         public VerticalMetrics Get1000EmGlyphVerticalMetrics(char glyph)
02406         {
02407             int num = (int)glyph;
02408             if (num < 256)
02409             {
02410                 return VerticalMetricsCache[num];
```

```
02411             }
02412             else
02413             {
02414                 return Get1000EmGlyphVerticalMetrics(GetGlyphIndex(glyph));
02415             }
02416         }
02417 
02418 /// <summary>
02419 /// Computes the distance of the top of the underline from the baseline, in thousandths of em unit.
02420 /// </summary>
02421 /// <returns>The distance of the top of the underline from the baseline, in thousandths of em
      unit.</returns>
02422         public double Get1000EmUnderlinePosition()
02423         {
02424             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02425             {
02426                 return post.UnderlinePosition * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02427             }
02428             else
02429             {
02430                 return double.NaN;
02431             }
02432         }
02433 
02434 /// <summary>
02435 /// Computes the thickness of the underline, in thousandths of em unit.
02436 /// </summary>
02437 /// <returns>The thickness of the underline, in thousandths of em unit.</returns>
02438         public double Get1000EmUnderlineThickness()
02439         {
02440             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02441             {
02442                 return post.UnderlineThickness * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02443             }
02444             else
02445             {
02446                 return double.NaN;
02447             }
02448         }
02449 
02450 /// <summary>
02451 /// Returns the number of units for each em in the font file.  Multiplying any glyph measurement by
      this
02452 /// size should generally lead to an integer value.
02453 /// </summary>
02454 /// <returns>The number of units for each em in the font file.</returns>
02455         public int GetUnitsPerEm()
02456         {
02457             return ((TrueTypeHeadTable)Tables["head"]).UnitsPerEm;
02458         }
02459 
02460 /// <summary>
02461 /// Computes the italic angle for the current font, in thousandths of em unit.  This is computed from
      the vertical and is negative for text that leans forwards.
02462 /// </summary>
02463 /// <returns></returns>
02464         public double GetItalicAngle()
02465         {
02466             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02467             {
02468                 return post.ItalicAngle.Bits / Math.Pow(2, post.ItalicAngle.BitShifts);
02469             }
02470             else
02471             {
02472                 return double.NaN;
02473             }
02474         }
02475 
02476 /// <summary>
02477 /// Computes the intersections between an underline at the specified position and thickness and a
      glyph, in thousandths of em units.
02478 /// </summary>
02479 /// <param name="glyph">The glyph whose intersections with the underline will be computed.</param>
02480 /// <param name="position">The distance of the top of the underline from the baseline, in thousandths
      of em unit.</param>
02481 /// <param name="thickness">The thickness of the underline, in thousandths of em unit.</param>
02482 /// <returns>If the underline does not intersect the glyph, this method returns <see langword="null"
      />.  Otherwise, it returns an array
02483 /// containing two elements, representing the horizontal coordinates of the leftmost and rightmost
      intersection points.</returns>
02484         public double[] Get1000EmUnderlineIntersections(char glyph, double position, double thickness)
02485         {
```

```
02486             List<double> intersections = new List<double>();
02487
02488             TrueTypePoint[][] glyphPaths = GetGlyphPath(glyph, 1000);
02489
02490             for (int j = 0; j < glyphPaths.Length; j++)
02491             {
02492                 double[] currPoint = new double[] { glyphPaths[j][0].X, -glyphPaths[j][0].Y };
02493
02494                 for (int k = 1; k < glyphPaths[j].Length; k++)
02495                 {
02496                     if (glyphPaths[j][k].IsOnCurve)
02497                     {
02498                         if (-glyphPaths[j][k].Y - currPoint[1] != 0)
02499                         {
02500                             double t = (position - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02501                             if (t >= 0 && t <= 1)
02502                             {
02503                                 intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02504                             }
02505
02506                             t = (position + thickness - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02507                             if (t >= 0 && t <= 1)
02508                             {
02509                                 intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02510                             }
02511                         }
02512                         else
02513                         {
02514                             if (position <= currPoint[1] && position + thickness >= currPoint[1])
02515                             {
02516                                 intersections.Add(currPoint[0]);
02517                                 intersections.Add(glyphPaths[j][k].X);
02518                             }
02519                         }
02520
02521                         currPoint = new double[] { glyphPaths[j][k].X, -glyphPaths[j][k].Y };
02522                     }
02523                     else
02524                     {
02525                         double[] ctrlPoint = new double[] { glyphPaths[j][k].X, -glyphPaths[j][k].Y };
02526                         double[] endPoint = new double[] { glyphPaths[j][k + 1].X, -glyphPaths[j][k +
      1].Y };
02527
02528                         double a = currPoint[1] - 2 * ctrlPoint[1] + endPoint[1];
02529
02530                         if (Math.Abs(a) < 1e-7)
02531                         {
02532                             k++;
02533
02534                             if (-glyphPaths[j][k].Y - currPoint[1] != 0)
02535                             {
02536                                 double t = (position - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02537                                 if (t >= 0 && t <= 1)
02538                                 {
02539                                     intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02540                                 }
02541
02542                                 t = (position + thickness - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02543                                 if (t >= 0 && t <= 1)
02544                                 {
02545                                     intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02546                                 }
02547                             }
02548                             else
02549                             {
02550                                 if (position <= currPoint[1] && position + thickness >= currPoint[1])
02551                                 {
02552                                     intersections.Add(currPoint[0]);
02553                                     intersections.Add(glyphPaths[j][k].X);
02554                                 }
02555                             }
02556                         }
02557                         else
02558                         {
02559                             double[] ts = new double[4];
02560
02561                             ts[0] = (currPoint[1] - ctrlPoint[1] - Math.Sqrt(position * a +
      ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02562                             ts[1] = (currPoint[1] - ctrlPoint[1] + Math.Sqrt(position * a +
```

```
      ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02563
02564                             ts[2] = (currPoint[1] - ctrlPoint[1] - Math.Sqrt((position + thickness) *
      a + ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02565                             ts[3] = (currPoint[1] - ctrlPoint[1] + Math.Sqrt((position + thickness) *
      a + ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02566
02567                             double minT = double.MaxValue;
02568                             double maxT = double.MinValue;
02569                             bool found = false;
02570
02571                             for (int i = 0; i < 4; i++)
02572                             {
02573                                 if (!double.IsNaN(ts[i]) && ts[i] >= 0 && ts[i] <= 1)
02574                                 {
02575                                     minT = Math.Min(minT, ts[i]);
02576                                     maxT = Math.Max(maxT, ts[i]);
02577
02578                                     found = true;
02579                                 }
02580                             }
02581
02582                             if (found)
02583                             {
02584                                 double critT = (currPoint[0] - ctrlPoint[0]) / (currPoint[0] - 2 *
      ctrlPoint[0] + endPoint[0]);
02585
02586                                 if (critT >= minT && critT <= maxT)
02587                                 {
02588                                     intersections.Add((1 - critT) * (1 - critT) * currPoint[0] + 2 *
      critT * (1 - critT) * ctrlPoint[0] + critT * critT * endPoint[0]);
02589                                 }
02590
02591                                 intersections.Add((1 - minT) * (1 - minT) * currPoint[0] + 2 * minT *
      (1 - minT) * ctrlPoint[0] + minT * minT * endPoint[0]);
02592                                 intersections.Add((1 - maxT) * (1 - maxT) * currPoint[0] + 2 * maxT *
      (1 - maxT) * ctrlPoint[0] + maxT * maxT * endPoint[0]);
02593                             }
02594
02595                             k++;
02596                         }
02597
02598                         currPoint = endPoint;
02599                     }
02600                 }
02601             }
02602
02603             if (intersections.Count > 1)
02604             {
02605                 double[] tbr = new double[2] { double.MaxValue, double.MinValue };
02606
02607                 for (int i = 0; i < intersections.Count; i++)
02608                 {
02609                     if (intersections[i] <= tbr[0])
02610                     {
02611                         tbr[0] = intersections[i];
02612                     }
02613
02614                     if (intersections[i] >= tbr[1])
02615                     {
02616                         tbr[1] = intersections[i];
02617                     }
02618                 }
02619
02620                 return tbr;
02621             }
02622             else
02623             {
02624                 return null;
02625             }
02626         }
02627
02628 /// <summary>
02629 /// Gets the kerning between two glyphs.
02630 /// </summary>
02631 /// <param name="glyph1">The first glyph of the kerning pair.</param>
02632 /// <param name="glyph2">The second glyph of the kerning pair.</param>
02633 /// <returns>A <see cref="PairKerning"/> object containing information about how the position of each
      glyphs should be altered.</returns>
02634         public PairKerning Get1000EmKerning(char glyph1, char glyph2)
02635         {
02636             int glyph1Index = this.GetGlyphIndex(glyph1);
02637             int glyph2Index = this.GetGlyphIndex(glyph2);
02638
02639             return Get1000EmKerning(glyph1Index, glyph2Index);
02640         }
02641
```

```
02642 /// <summary>
02643 /// Gets the kerning between two glyphs.
02644 /// </summary>
02645 /// <param name="glyph1Index">The index of the first glyph of the kerning pair.</param>
02646 /// <param name="glyph2Index">The index of the second glyph of the kerning pair.</param>
02647 /// <returns>A <see cref="PairKerning"/> object containing information about how the position of each
      glyphs should be altered.</returns>
02648         public PairKerning Get1000EmKerning(int glyph1Index, int glyph2Index)
02649         {
02650             if (this.Tables.TryGetValue("GPOS", out ITrueTypeTable table) && table is
      TrueTypeGPOSTable gpos)
02651             {
02652                 PairKerning kerning = gpos.GetKerning(glyph1Index, glyph2Index);
02653 
02654                 if (kerning != null)
02655                 {
02656                     double units = ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02657 
02658                     return new PairKerning(new Point(kerning.Glyph1Placement.X * 1000 / units,
      kerning.Glyph1Placement.Y * 1000 / units),
02659                         new Point(kerning.Glyph1Advance.X * 1000 / units, kerning.Glyph1Advance.Y *
      1000 / units),
02660                         new Point(kerning.Glyph2Placement.X * 1000 / units, kerning.Glyph2Placement.Y
      * 1000 / units),
02661                         new Point(kerning.Glyph2Advance.X * 1000 / units, kerning.Glyph2Advance.Y *
      1000 / units));
02662                 }
02663                 else
02664                 {
02665                     return null;
02666                 }
02667             }
02668             else
02669             {
02670                 return null;
02671             }
02672         }
02673 
02674         internal interface ITrueTypeTable
02675         {
02676             byte[] GetBytes();
02677         }
02678 
02679         internal interface ICmapTable
02680         {
02681             ushort Format { get; }
02682             ushort Length { get; }
02683             ushort Language { get; }
02684 
02685             int GetGlyphIndex(char glyph);
02686 
02687             byte[] GetBytes();
02688         }
02689 
02690         internal class CmapTable4 :  ICmapTable
02691         {
02692             public ushort Format { get; set; }
02693             public ushort Length { get; set; }
02694             public ushort Language { get; set; }
02695             public ushort SegCountX2 { get; set; }
02696             public ushort SearchRange { get; set; }
02697             public ushort EntrySelector { get; set; }
02698             public ushort RangeShift { get; set; }
02699             public ushort[] EndCode { get; set; }
02700             public ushort ReservedPad { get; set; }
02701             public ushort[] StartCode { get; set; }
02702             public ushort[] IdDelta { get; set; }
02703             public ushort[] IdRangeOffset { get; set; }
02704             public ushort[] GlyphIndexArray { get; set; }
02705 
02706             public CmapTable4() { }
02707 
02708             public CmapTable4(ushort format, ushort length, ushort language, Stream sr)
02709             {
02710                 this.Format = format;
02711                 this.Length = length;
02712                 this.Language = language;
02713                 this.SegCountX2 = sr.ReadUShort();
02714 
02715                 int segCount = this.SegCountX2 / 2;
02716 
02717                 this.SearchRange = sr.ReadUShort();
02718                 this.EntrySelector = sr.ReadUShort();
02719                 this.RangeShift = sr.ReadUShort();
02720 
02721                 this.EndCode = new ushort[segCount];
02722                 for (int i = 0; i < segCount; i++)
```

```
02723                 {
02724                     this.EndCode[i] = sr.ReadUShort();
02725                 }
02726 
02727                 this.ReservedPad = sr.ReadUShort();
02728 
02729                 this.StartCode = new ushort[segCount];
02730                 for (int i = 0; i < segCount; i++)
02731                 {
02732                     this.StartCode[i] = sr.ReadUShort();
02733                 }
02734 
02735                 this.IdDelta = new ushort[segCount];
02736                 for (int i = 0; i < segCount; i++)
02737                 {
02738                     this.IdDelta[i] = sr.ReadUShort();
02739                 }
02740 
02741                 this.IdRangeOffset = new ushort[segCount];
02742                 for (int i = 0; i < segCount; i++)
02743                 {
02744                     this.IdRangeOffset[i] = sr.ReadUShort();
02745                 }
02746 
02747                 int numGlyphIndices = (this.Length - 16 + 8 * segCount) / 2;
02748 
02749                 this.GlyphIndexArray = new ushort[numGlyphIndices];
02750 
02751                 for (int i = 0; i < numGlyphIndices; i++)
02752                 {
02753                     this.GlyphIndexArray[i] = sr.ReadUShort();
02754                 }
02755             }
02756 
02757             public int GetGlyphIndex(char glyph)
02758             {
02759                 int code = (int)glyph;
02760 
02761                 int endCodeInd = -1;
02762 
02763                 for (int i = 0; i < EndCode.Length; i++)
02764                 {
02765                     if (EndCode[i] >= code)
02766                     {
02767                         endCodeInd = i;
02768                         break;
02769                     }
02770                 }
02771 
02772                 if (StartCode[endCodeInd] <= code)
02773                 {
02774                     if (IdRangeOffset[endCodeInd] != 0)
02775                     {
02776                         int glyphIndexIndex = IdRangeOffset[endCodeInd] / 2 + (code -
    StartCode[endCodeInd]) - (IdRangeOffset.Length - endCodeInd);
02777 
02778                         if (GlyphIndexArray[glyphIndexIndex] != 0)
02779                         {
02780                             return (IdDelta[endCodeInd] + GlyphIndexArray[glyphIndexIndex]) % 65536;
02781                         }
02782                         else
02783                         {
02784                             return 0;
02785                         }
02786                     }
02787                     else
02788                     {
02789                         return (code + IdDelta[endCodeInd]) % 65536;
02790                     }
02791                 }
02792                 else
02793                 {
02794                     return 0;
02795                 }
02796             }
02797 
02798             public byte[] GetBytes()
02799             {
02800                 using (MemoryStream ms = new MemoryStream())
02801                 {
02802                     ms.WriteUShort(this.Format);
02803                     ms.WriteUShort(this.Length);
02804                     ms.WriteUShort(this.Language);
02805                     ms.WriteUShort(this.SegCountX2);
02806                     ms.WriteUShort(this.SearchRange);
02807                     ms.WriteUShort(this.EntrySelector);
02808                     ms.WriteUShort(this.RangeShift);
```

```
02809                     for (int i = 0; i < this.EndCode.Length; i++)
02810                     {
02811                         ms.WriteUShort(this.EndCode[i]);
02812                     }
02813                     ms.WriteUShort(this.ReservedPad);
02814 
02815                     for (int i = 0; i < this.StartCode.Length; i++)
02816                     {
02817                         ms.WriteUShort(this.StartCode[i]);
02818                     }
02819                     for (int i = 0; i < this.IdDelta.Length; i++)
02820                     {
02821                         ms.WriteUShort(this.IdDelta[i]);
02822                     }
02823                     for (int i = 0; i < this.IdRangeOffset.Length; i++)
02824                     {
02825                         ms.WriteUShort(this.IdRangeOffset[i]);
02826                     }
02827                     for (int i = 0; i < this.GlyphIndexArray.Length; i++)
02828                     {
02829                         ms.WriteUShort(this.GlyphIndexArray[i]);
02830                     }
02831 
02832                     return ms.ToArray();
02833                 }
02834             }
02835         }
02836 
02837         internal class CmapTable0 :  ICmapTable
02838         {
02839             public ushort Format { get; }
02840             public ushort Length { get; }
02841             public ushort Language { get; }
02842 
02843             public byte[] GlyphIndexArray { get; }
02844 
02845             public CmapTable0(ushort format, ushort length, ushort language, byte[] glyphIndexArray)
02846             {
02847                 this.Format = format;
02848                 this.Length = length;
02849                 this.Language = language;
02850                 this.GlyphIndexArray = glyphIndexArray;
02851             }
02852 
02853             public int GetGlyphIndex(char glyph)
02854             {
02855                 return GlyphIndexArray[(byte)glyph];
02856             }
02857 
02858             public byte[] GetBytes()
02859             {
02860                 using (MemoryStream ms = new MemoryStream())
02861                 {
02862                     ms.WriteUShort(this.Format);
02863                     ms.WriteUShort(this.Length);
02864                     ms.WriteUShort(this.Language);
02865                     ms.Write(this.GlyphIndexArray, 0, this.GlyphIndexArray.Length);
02866                     return ms.ToArray();
02867                 }
02868             }
02869         }
02870 
02871         internal struct CmapSubTable
02872         {
02873             public ushort PlatformID;
02874             public ushort PlatformSpecificID;
02875             public uint Offset;
02876 
02877             public CmapSubTable(ushort platformID, ushort platformSpecificID, uint offset)
02878             {
02879                 this.PlatformID = platformID;
02880                 this.PlatformSpecificID = platformSpecificID;
02881                 this.Offset = offset;
02882             }
02883         }
02884 
02885         internal class TrueTypeCmapTable :  ITrueTypeTable
02886         {
02887             public ushort Version;
02888             public ushort NumberSubTables;
02889             public CmapSubTable[] SubTables;
02890             public ICmapTable[] ActualCmapTables;
02891 
02892             public byte[] GetBytes()
02893             {
02894                 using (MemoryStream ms = new MemoryStream())
02895                 {
```

```
02896                     ms.WriteUShort(this.Version);
02897                     ms.WriteUShort(this.NumberSubTables);
02898 
02899                     for (int i = 0; i < this.SubTables.Length; i++)
02900                     {
02901                         ms.WriteUShort(this.SubTables[i].PlatformID);
02902                         ms.WriteUShort(this.SubTables[i].PlatformSpecificID);
02903                         ms.WriteUInt(this.SubTables[i].Offset);
02904                     }
02905 
02906                     for (int i = 0; i < this.ActualCmapTables.Length; i++)
02907                     {
02908                         byte[] bytes = this.ActualCmapTables[i].GetBytes();
02909                         ms.Write(bytes, 0, bytes.Length);
02910                     }
02911 
02912                     return ms.ToArray();
02913                 }
02914             }
02915         }
02916 
02917         internal class TrueTypeNameTable :  ITrueTypeTable
02918         {
02919             public byte[] GetBytes()
02920             {
02921                 using (MemoryStream ms = new MemoryStream())
02922                 {
02923                     ms.WriteUShort(this.Format);
02924                     ms.WriteUShort(this.Count);
02925                     ms.WriteUShort(this.StringOffset);
02926 
02927                     for (int i = 0; i < this.NameRecords.Length; i++)
02928                     {
02929                         byte[] bytes = this.NameRecords[i].GetBytes();
02930                         ms.Write(bytes, 0, bytes.Length);
02931                     }
02932 
02933                     ms.Write(this.RawBytes, 0, this.RawBytes.Length);
02934 
02935                     return ms.ToArray();
02936                 }
02937             }
02938 
02939             public ushort Format;
02940             public ushort Count;
02941             public ushort StringOffset;
02942             public NameRecord[] NameRecords;
02943             public string[] Name;
02944             private readonly byte[] RawBytes;
02945 
02946             internal struct NameRecord
02947             {
02948                 public ushort PlatformID;
02949                 public ushort PlatformSpecificID;
02950                 public ushort LanguageID;
02951                 public ushort NameID;
02952                 public ushort Length;
02953                 public ushort Offset;
02954 
02955                 public NameRecord(ushort platformID, ushort platformSpecificID, ushort languageID,
    ushort nameID, ushort length, ushort offset)
02956                 {
02957                     this.PlatformID = platformID;
02958                     this.PlatformSpecificID = platformSpecificID;
02959                     this.LanguageID = languageID;
02960                     this.NameID = nameID;
02961                     this.Length = length;
02962                     this.Offset = offset;
02963                 }
02964 
02965                 public byte[] GetBytes()
02966                 {
02967                     using (MemoryStream ms = new MemoryStream())
02968                     {
02969                         ms.WriteUShort(this.PlatformID);
02970                         ms.WriteUShort(this.PlatformSpecificID);
02971                         ms.WriteUShort(this.LanguageID);
02972                         ms.WriteUShort(this.NameID);
02973                         ms.WriteUShort(this.Length);
02974                         ms.WriteUShort(this.Offset);
02975                         return ms.ToArray();
02976                     }
02977                 }
02978             }
02979 
02980             internal TrueTypeNameTable(uint tableOffset, Stream sr)
02981             {
```

```
02982                 this.Format = sr.ReadUShort();
02983                 this.Count = sr.ReadUShort();
02984                 this.StringOffset = sr.ReadUShort();
02985                 this.NameRecords = new NameRecord[this.Count];
02986                 this.Name = new string[this.Count];
02987 
02988                 int maxOffsetItem = -1;
02989 
02990                 for (int i = 0; i < this.Count; i++)
02991                 {
02992                     this.NameRecords[i] = new NameRecord(sr.ReadUShort(), sr.ReadUShort(),
      sr.ReadUShort(), sr.ReadUShort(), sr.ReadUShort(), sr.ReadUShort());
02993                     if (maxOffsetItem < 0 || this.NameRecords[i].Offset >
      this.NameRecords[maxOffsetItem].Offset || (this.NameRecords[i].Offset ==
      this.NameRecords[maxOffsetItem].Offset && this.NameRecords[i].Length >
      this.NameRecords[maxOffsetItem].Length))
02994                     {
02995                         maxOffsetItem = i;
02996                     }
02997                 }
02998 
02999                 this.RawBytes = new byte[this.NameRecords[maxOffsetItem].Offset +
      this.NameRecords[maxOffsetItem].Length];
03000                 sr.Seek(tableOffset + this.StringOffset, SeekOrigin.Begin);
03001                 sr.Read(this.RawBytes, 0, this.RawBytes.Length);
03002 
03003                 for (int i = 0; i < this.Count; i++)
03004                 {
03005                     sr.Seek(tableOffset + this.NameRecords[i].Offset + this.StringOffset,
      SeekOrigin.Begin);
03006                     byte[] stringBytes = new byte[this.NameRecords[i].Length];
03007                     sr.Read(stringBytes, 0, this.NameRecords[i].Length);
03008 
03009                     if (this.NameRecords[i].PlatformID == 0)
03010                     {
03011                         this.Name[i] = Encoding.BigEndianUnicode.GetString(stringBytes);
03012                     }
03013                     else if (this.NameRecords[i].PlatformID == 1 &&
      this.NameRecords[i].PlatformSpecificID == 0)
03014                     {
03015                         this.Name[i] = GetMacRomanString(stringBytes);
03016                     }
03017                     else if (this.NameRecords[i].PlatformID == 3 &&
      (this.NameRecords[i].PlatformSpecificID == 1 || this.NameRecords[i].PlatformSpecificID == 0))
03018                     {
03019                         this.Name[i] = Encoding.BigEndianUnicode.GetString(stringBytes);
03020                     }
03021                     else
03022                     {
03023                         this.Name[i] = "Unsupported encoding:  " +
      this.NameRecords[i].PlatformID.ToString() + "/" + this.NameRecords[i].PlatformSpecificID.ToString();
03024                     }
03025 
03026                 }
03027             }
03028 
03029             private static readonly char[] MacRomanChars = new char[] { '\u0020', '\u0021', '\u0022',
      '\u0023', '\u0024', '\u0025', '\u0026', '\u0027', '\u0028', '\u0029', '\u002a', '\u002b', '\u002c',
      '\u002d', '\u002e', '\u002f', '\u0030', '\u0031', '\u0032', '\u0033', '\u0034', '\u0035', '\u0036',
      '\u0037', '\u0038', '\u0039', '\u003a', '\u003b', '\u003c', '\u003d', '\u003e', '\u003f', '\u0040',
      '\u0041', '\u0042', '\u0043', '\u0044', '\u0045', '\u0046', '\u0047', '\u0048', '\u0049', '\u004a',
      '\u004b', '\u004c', '\u004d', '\u004e', '\u004f', '\u0050', '\u0051', '\u0052', '\u0053', '\u0054',
      '\u0055', '\u0056', '\u0057', '\u0058', '\u0059', '\u005a', '\u005b', '\u005c', '\u005d', '\u005e',
      '\u005f', '\u0060', '\u0061', '\u0062', '\u0063', '\u0064', '\u0065', '\u0066', '\u0067', '\u0068',
      '\u0069', '\u006a', '\u006b', '\u006c', '\u006d', '\u006e', '\u006f', '\u0070', '\u0071', '\u0072',
      '\u0073', '\u0074', '\u0075', '\u0076', '\u0077', '\u0078', '\u0079', '\u007a', '\u007b', '\u007c',
      '\u007d', '\u007e', '\u007f', '\u00c4', '\u00c5', '\u00c7', '\u00c9', '\u00d1', '\u00d6', '\u00dc',
      '\u00e1', '\u00e0', '\u00e2', '\u00e4', '\u00e3', '\u00e5', '\u00e7', '\u00e9', '\u00e8', '\u00ea',
      '\u00eb', '\u00ed', '\u00ec', '\u00ee', '\u00ef', '\u00f1', '\u00f3', '\u00f2', '\u00f4', '\u00f6',
      '\u00f5', '\u00fa', '\u00f9', '\u00fb', '\u00fc', '\u2020', '\u00b0', '\u00a2', '\u00a3', '\u00a7',
      '\u2022', '\u00b6', '\u00df', '\u00ae', '\u00a9', '\u2122', '\u00b4', '\u00a8', '\u2260', '\u00c6',
      '\u00d8', '\u221e', '\u00b1', '\u2264', '\u2265', '\u00a5', '\u00b5', '\u2202', '\u2211', '\u220f',
      '\u03c0', '\u222b', '\u00aa', '\u00ba', '\u03a9', '\u00e6', '\u00f8', '\u00bf', '\u00a1', '\u00ac',
      '\u221a', '\u0192', '\u2248', '\u2206', '\u00ab', '\u00bb', '\u2026', '\u00a0', '\u00c0', '\u00c3',
      '\u00d5', '\u0152', '\u0153', '\u2013', '\u2014', '\u201c', '\u201d', '\u2018', '\u2019', '\u00f7',
      '\u25ca', '\u00ff', '\u0178', '\u2044', '\u20ac', '\u2039', '\u203a', '\ufb01', '\ufb02', '\u2021',
      '\u00b7', '\u201a', '\u201e', '\u2030', '\u00c2', '\u00ca', '\u00c1', '\u00cb', '\u00c8', '\u00cd',
      '\u00ce', '\u00cf', '\u00cc', '\u00d3', '\u00d4', '\uf8ff', '\u00d2', '\u00da', '\u00db', '\u00d9',
      '\u0131', '\u02c6', '\u02dc', '\u00af', '\u02d8', '\u02d9', '\u02da', '\u00b8', '\u02dd', '\u02db',
      '\u02c7' };
03030 
03031             private static string GetMacRomanString(byte[] bytes)
03032             {
03033                 StringBuilder bld = new StringBuilder(bytes.Length);
03034 
03035                 for (int i = 0; i < bytes.Length; i++)
03036                 {
```

```
03037                   if (bytes[i] >= 32)
03038                   {
03039                       bld.Append(MacRomanChars[bytes[i] - 32]);
03040                   }
03041                   else
03042                   {
03043                       bld.Append((char)bytes[i]);
03044                   }
03045               }
03046
03047               return bld.ToString();
03048
03049           }
03050       }
03051
03052       internal class TrueTypeOS2Table :  ITrueTypeTable
03053       {
03054           public ushort Version;
03055           public short XAvgCharWidth;
03056           public ushort UsWeightClass;
03057           public ushort UsWidthClass;
03058           public short FsType;
03059           public short YSubscriptXSize;
03060           public short YSubscriptYSize;
03061           public short YSubscriptXOffset;
03062           public short YSubscriptYOffset;
03063           public short YSuperscriptXSize;
03064           public short YSuperscriptYSize;
03065           public short YSuperscriptXOffset;
03066           public short YSuperscriptYOffset;
03067           public short YStrikeoutSize;
03068           public short YStrikeoutPosition;
03069           public byte SFamilyClass;
03070           public byte SFamilySubClass;
03071           public PANOSE Panose;
03072           public uint[] UlUnicodeRange;
03073           public byte[] AchVendID;
03074           public ushort FsSelection;
03075           public ushort FsFirstCharIndex;
03076           public ushort FsLastCharIndex;
03077           public short STypoAscender;
03078           public short STypoDescender;
03079           public short STypoLineGap;
03080           public ushort UsWinAscent;
03081           public ushort UsWinDescent;
03082
03083           public uint[] UlCodePageRange;
03084
03085           public short SxHeight;
03086           public short SCapHeight;
03087           public ushort UsDefaultChar;
03088           public ushort UsBreakChar;
03089           public ushort UsMaxContext;
03090
03091           public ushort UsLowerOpticalPointSize;
03092           public ushort UsUpperOpticalPointSize;
03093
03094           public byte[] GetBytes()
03095           {
03096               using (MemoryStream ms = new MemoryStream())
03097               {
03098                   ms.WriteUShort(this.Version);
03099                   ms.WriteShort(this.XAvgCharWidth);
03100                   ms.WriteUShort(this.UsWeightClass);
03101                   ms.WriteUShort(this.UsWidthClass);
03102                   ms.WriteShort(this.FsType);
03103                   ms.WriteShort(this.YSubscriptXSize);
03104                   ms.WriteShort(this.YSubscriptYSize);
03105                   ms.WriteShort(this.YSubscriptXOffset);
03106                   ms.WriteShort(this.YSubscriptYOffset);
03107                   ms.WriteShort(this.YSuperscriptXSize);
03108                   ms.WriteShort(this.YSuperscriptYSize);
03109                   ms.WriteShort(this.YSuperscriptXOffset);
03110                   ms.WriteShort(this.YSuperscriptYOffset);
03111                   ms.WriteShort(this.YStrikeoutSize);
03112                   ms.WriteShort(this.YStrikeoutPosition);
03113                   ms.WriteByte(this.SFamilyClass);
03114                   ms.WriteByte(this.SFamilySubClass);
03115                   ms.Write(this.Panose.GetBytes(), 0, 10);
03116                   for (int i = 0; i < this.UlUnicodeRange.Length; i++)
03117                   {
03118                       ms.WriteUInt(this.UlUnicodeRange[i]);
03119                   }
03120                   ms.Write(this.AchVendID, 0, this.AchVendID.Length);
03121                   ms.WriteUShort(this.FsSelection);
03122                   ms.WriteUShort(this.FsFirstCharIndex);
03123                   ms.WriteUShort(this.FsLastCharIndex);
```

```
03124                     ms.WriteShort(this.STypoAscender);
03125                     ms.WriteShort(this.STypoDescender);
03126                     ms.WriteShort(this.STypoLineGap);
03127                     ms.WriteUShort(this.UsWinAscent);
03128                     ms.WriteUShort(this.UsWinDescent);
03129 
03130                     if (this.Version >= 1)
03131                     {
03132                         for (int i = 0; i < this.UlCodePageRange.Length; i++)
03133                         {
03134                             ms.WriteUInt(this.UlCodePageRange[i]);
03135                         }
03136 
03137                         if (this.Version >= 2)
03138                         {
03139                             ms.WriteShort(this.SxHeight);
03140                             ms.WriteShort(this.SCapHeight);
03141                             ms.WriteUShort(this.UsDefaultChar);
03142                             ms.WriteUShort(this.UsBreakChar);
03143                             ms.WriteUShort(this.UsMaxContext);
03144 
03145                             if (this.Version >= 5)
03146                             {
03147                                 ms.WriteUShort(this.UsLowerOpticalPointSize);
03148                                 ms.WriteUShort(this.UsUpperOpticalPointSize);
03149                             }
03150                         }
03151                     }
03152 
03153                     return ms.ToArray();
03154                 }
03155             }
03156 
03157             internal struct PANOSE
03158             {
03159                 public byte BFamilyType;
03160                 public byte BSerifStyle;
03161                 public byte BWeight;
03162                 public byte BProportion;
03163                 public byte BContrast;
03164                 public byte BStrokeVariation;
03165                 public byte BArmStyle;
03166                 public byte BLetterform;
03167                 public byte BMidline;
03168                 public byte BXHeight;
03169 
03170                 public PANOSE(byte bFamilyType, byte bSerifStyle, byte bWeight, byte bProportion, byte
      bContrast, byte bStrokeVariation, byte bArmStyle, byte bLetterform, byte bMidline, byte bXHeight)
03171                 {
03172                     this.BFamilyType = bFamilyType;
03173                     this.BSerifStyle = bSerifStyle;
03174                     this.BWeight = bWeight;
03175                     this.BProportion = bProportion;
03176                     this.BContrast = bContrast;
03177                     this.BStrokeVariation = bStrokeVariation;
03178                     this.BArmStyle = bArmStyle;
03179                     this.BLetterform = bLetterform;
03180                     this.BMidline = bMidline;
03181                     this.BXHeight = bXHeight;
03182                 }
03183 
03184                 public byte[] GetBytes()
03185                 {
03186                     return new byte[] { BFamilyType, BSerifStyle, BWeight, BProportion, BContrast,
      BStrokeVariation, BArmStyle, BLetterform, BMidline, BXHeight };
03187                 }
03188             }
03189         }
03190 
03191         internal class TrueTypePostTable :  ITrueTypeTable
03192         {
03193             public Fixed Version;
03194             public Fixed ItalicAngle;
03195             public short UnderlinePosition;
03196             public short UnderlineThickness;
03197             public uint IsFixedPitch;
03198             public uint MinMemType42;
03199             public uint MaxMemType42;
03200             public uint MinMemType1;
03201             public uint MaxMemType1;
03202 
03203             public ushort NumGlyphs;
03204             public ushort[] GlyphNameIndex;
03205             public string[] Names;
03206 
03207             public byte[] GetBytes()
03208             {
```

```
03209                 using (MemoryStream ms = new MemoryStream())
03210                 {
03211                     ms.WriteFixed(this.Version);
03212                     ms.WriteFixed(this.ItalicAngle);
03213                     ms.WriteShort(this.UnderlinePosition);
03214                     ms.WriteShort(this.UnderlineThickness);
03215                     ms.WriteUInt(this.IsFixedPitch);
03216                     ms.WriteUInt(this.MinMemType42);
03217                     ms.WriteUInt(this.MaxMemType42);
03218                     ms.WriteUInt(this.MinMemType1);
03219                     ms.WriteUInt(this.MaxMemType1);
03220 
03221                     if (this.Version.Bits == 0x00020000)
03222                     {
03223                         ms.WriteUShort(NumGlyphs);
03224                         for (int i = 0; i < GlyphNameIndex.Length; i++)
03225                         {
03226                             ms.WriteUShort(GlyphNameIndex[i]);
03227                         }
03228 
03229                         for (int i = 0; i < Names.Length; i++)
03230                         {
03231                             ms.WritePascalString(Names[i]);
03232                         }
03233                     }
03234 
03235                     return ms.ToArray();
03236                 }
03237             }
03238         }
03239 
03240         internal class TrueTypeHmtxTable :  ITrueTypeTable
03241         {
03242             public LongHorFixed[] HMetrics;
03243             public short[] LeftSideBearing;
03244 
03245             public byte[] GetBytes()
03246             {
03247                 using (MemoryStream ms = new MemoryStream())
03248                 {
03249                     for (int i = 0; i < HMetrics.Length; i++)
03250                     {
03251                         ms.WriteUShort(HMetrics[i].AdvanceWidth);
03252                         ms.WriteShort(HMetrics[i].LeftSideBearing);
03253                     }
03254                     for (int i = 0; i < LeftSideBearing.Length; i++)
03255                     {
03256                         ms.WriteShort(LeftSideBearing[i]);
03257                     }
03258 
03259                     return ms.ToArray();
03260                 }
03261             }
03262         }
03263 
03264         internal class TrueTypeMaxpTable :  ITrueTypeTable
03265         {
03266             public Fixed Version;
03267             public ushort NumGlyphs;
03268             public ushort MaxPoints;
03269             public ushort MaxContours;
03270             public ushort MaxComponentPoints;
03271             public ushort MaxComponentContours;
03272             public ushort MaxZones;
03273             public ushort MaxTwilightPoints;
03274             public ushort MaxStorage;
03275             public ushort MaxFunctionDefs;
03276             public ushort MaxInstructionDefs;
03277             public ushort MaxStackElements;
03278             public ushort MaxSizeOfInstructions;
03279             public ushort MaxComponentElements;
03280             public ushort MaxComponentDepth;
03281 
03282             public byte[] GetBytes()
03283             {
03284                 using (MemoryStream ms = new MemoryStream())
03285                 {
03286                     ms.WriteFixed(this.Version);
03287                     ms.WriteUShort(this.NumGlyphs);
03288                     ms.WriteUShort(this.MaxPoints);
03289                     ms.WriteUShort(this.MaxContours);
03290                     ms.WriteUShort(this.MaxComponentPoints);
03291                     ms.WriteUShort(this.MaxComponentContours);
03292                     ms.WriteUShort(this.MaxZones);
03293                     ms.WriteUShort(this.MaxTwilightPoints);
03294                     ms.WriteUShort(this.MaxStorage);
03295                     ms.WriteUShort(this.MaxFunctionDefs);
```

```
03296                     ms.WriteUShort(this.MaxInstructionDefs);
03297                     ms.WriteUShort(this.MaxStackElements);
03298                     ms.WriteUShort(this.MaxSizeOfInstructions);
03299                     ms.WriteUShort(this.MaxComponentElements);
03300                     ms.WriteUShort(this.MaxComponentDepth);
03301                     return ms.ToArray();
03302                 }
03303             }
03304         }
03305 
03306         internal class TrueTypeHeadTable :  ITrueTypeTable
03307         {
03308             public Fixed Version;
03309             public Fixed FontRevision;
03310             public uint ChecksumAdjustment;
03311             public uint MagicNumber;
03312             public ushort Flags;
03313             public ushort UnitsPerEm;
03314             public DateTime Created;
03315             public DateTime Modified;
03316             public short XMin;
03317             public short YMin;
03318             public short XMax;
03319             public short YMax;
03320             public ushort MacStyle;
03321             public ushort LowestRecPPEM;
03322             public short FontDirectionInt;
03323             public short IndexToLocFormat;
03324             public short GlyphDataFormat;
03325 
03326             public byte[] GetBytes()
03327             {
03328                 using (MemoryStream ms = new MemoryStream())
03329                 {
03330                     ms.WriteFixed(this.Version);
03331                     ms.WriteFixed(this.FontRevision);
03332                     ms.WriteUInt(this.ChecksumAdjustment);
03333                     ms.WriteUInt(this.MagicNumber);
03334                     ms.WriteUShort(this.Flags);
03335                     ms.WriteUShort(this.UnitsPerEm);
03336                     ms.WriteDate(this.Created);
03337                     ms.WriteDate(this.Modified);
03338                     ms.WriteShort(this.XMin);
03339                     ms.WriteShort(this.YMin);
03340                     ms.WriteShort(this.XMax);
03341                     ms.WriteShort(this.YMax);
03342                     ms.WriteUShort(this.MacStyle);
03343                     ms.WriteUShort(this.LowestRecPPEM);
03344                     ms.WriteShort(this.FontDirectionInt);
03345                     ms.WriteShort(this.IndexToLocFormat);
03346                     ms.WriteShort(this.GlyphDataFormat);
03347                     return ms.ToArray();
03348                 }
03349             }
03350 
03351         }
03352 
03353         internal class TrueTypeHHeaTable :  ITrueTypeTable
03354         {
03355             public Fixed Version;
03356             public short Ascent;
03357             public short Descent;
03358             public short LineGap;
03359             public ushort AdvanceWidthMax;
03360             public short MinLeftSideBearing;
03361             public short MinRightSideBearing;
03362             public short XMaxExtent;
03363             public short CaretSlopeRise;
03364             public short CaretSlopeRun;
03365             public short CaretOffset;
03366             public short MetricDataFormat;
03367             public ushort NumOfLongHorMetrics;
03368 
03369             public byte[] GetBytes()
03370             {
03371                 using (MemoryStream ms = new MemoryStream())
03372                 {
03373                     ms.WriteFixed(this.Version);
03374                     ms.WriteShort(this.Ascent);
03375                     ms.WriteShort(this.Descent);
03376                     ms.WriteShort(this.LineGap);
03377                     ms.WriteUShort(this.AdvanceWidthMax);
03378                     ms.WriteShort(this.MinLeftSideBearing);
03379                     ms.WriteShort(this.MinRightSideBearing);
03380                     ms.WriteShort(this.XMaxExtent);
03381                     ms.WriteShort(this.CaretSlopeRise);
03382                     ms.WriteShort(this.CaretSlopeRun);
```

```
03383                   ms.WriteShort(this.CaretOffset);
03384                   ms.WriteShort(0);
03385                   ms.WriteShort(0);
03386                   ms.WriteShort(0);
03387                   ms.WriteShort(0);
03388                   ms.WriteShort(this.MetricDataFormat);
03389                   ms.WriteUShort(this.NumOfLongHorMetrics);
03390                   return ms.ToArray();
03391               }
03392           }
03393       }
03394
03395
03396       internal class LangSysTable
03397       {
03398           public string Tag;
03399           public ushort LookupOrderOffset;
03400           public ushort RequiredFeatureIndex;
03401           public ushort[] FeatureIndices;
03402
03403           public LangSysTable(string tag, Stream fs)
03404           {
03405               this.Tag = tag;
03406               this.LookupOrderOffset = fs.ReadUShort();
03407               this.RequiredFeatureIndex = fs.ReadUShort();
03408               ushort featureIndexCount = fs.ReadUShort();
03409
03410               this.FeatureIndices = new ushort[featureIndexCount];
03411
03412               for (int i = 0; i < featureIndexCount; i++)
03413               {
03414                   this.FeatureIndices[i] = fs.ReadUShort();
03415               }
03416           }
03417
03418           public byte[] GetBytes()
03419           {
03420               using (MemoryStream ms = new MemoryStream(6 + this.FeatureIndices.Length * 2))
03421               {
03422                   ms.WriteUShort(this.LookupOrderOffset);
03423                   ms.WriteUShort(this.RequiredFeatureIndex);
03424                   ms.WriteUShort((ushort)this.FeatureIndices.Length);
03425                   for (int i = 0; i < this.FeatureIndices.Length; i++)
03426                   {
03427                       ms.WriteUShort(this.FeatureIndices[i]);
03428                   }
03429
03430                   return ms.ToArray();
03431               }
03432           }
03433       }
03434
03435       internal class ScriptTable
03436       {
03437           public string Tag;
03438           public LangSysTable DefaultLangSys;
03439           public LangSysTable[] LangSysRecords;
03440
03441           public ScriptTable(string tag, Stream fs)
03442           {
03443               this.Tag = tag;
03444
03445               long startPosition = fs.Position;
03446
03447               ushort defaultLangSysOffset = fs.ReadUShort();
03448               ushort langSysCount = fs.ReadUShort();
03449
03450               (string, ushort)[] langSysRecords = new (string, ushort)[langSysCount];
03451
03452               for (int i = 0; i < langSysCount; i++)
03453               {
03454                   StringBuilder langSysTag = new StringBuilder(4);
03455                   langSysTag.Append((char)fs.ReadByte());
03456                   langSysTag.Append((char)fs.ReadByte());
03457                   langSysTag.Append((char)fs.ReadByte());
03458                   langSysTag.Append((char)fs.ReadByte());
03459
03460                   langSysRecords[i] = (langSysTag.ToString(), fs.ReadUShort());
03461               }
03462
03463               if (defaultLangSysOffset != 0)
03464               {
03465                   fs.Seek(startPosition + defaultLangSysOffset, SeekOrigin.Begin);
03466                   this.DefaultLangSys = new LangSysTable(null, fs);
03467               }
03468               else
03469               {
```

```
03470                     this.DefaultLangSys = null;
03471                 }
03472 
03473                 this.LangSysRecords = new LangSysTable[langSysCount];
03474 
03475                 for (int i = 0; i < langSysCount; i++)
03476                 {
03477                     fs.Seek(startPosition + langSysRecords[i].Item2, SeekOrigin.Begin);
03478                     this.LangSysRecords[i] = new LangSysTable(langSysRecords[i].Item1, fs);
03479                 }
03480             }
03481 
03482             public byte[] GetBytes()
03483             {
03484                 using (MemoryStream ms = new MemoryStream())
03485                 {
03486                     int offset = 6 * this.LangSysRecords.Length + 2 + 2;
03487 
03488                     if (this.DefaultLangSys != null)
03489                     {
03490                         ms.WriteUShort((ushort)offset);
03491                     }
03492                     else
03493                     {
03494                         ms.WriteUShort(0);
03495                     }
03496 
03497                     ms.WriteUShort((ushort)this.LangSysRecords.Length);
03498 
03499                     byte[] defaultLangSysTable = DefaultLangSys != null ?  DefaultLangSys.GetBytes() :
    new byte[0];
03500 
03501                     byte[][] langSysTables = new byte[this.LangSysRecords.Length][];
03502 
03503                     offset += defaultLangSysTable.Length;
03504 
03505                     int[] offsets = new int[this.LangSysRecords.Length];
03506 
03507                     for (int i = 0; i < this.LangSysRecords.Length; i++)
03508                     {
03509                         offsets[i] = offset;
03510                         langSysTables[i] = this.LangSysRecords[i].GetBytes();
03511                         offset += langSysTables[i].Length;
03512                     }
03513 
03514                     for (int i = 0; i < this.LangSysRecords.Length; i++)
03515                     {
03516                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[0]);
03517                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[1]);
03518                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[2]);
03519                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[3]);
03520 
03521                         ms.WriteUShort((ushort)offsets[i]);
03522                     }
03523 
03524                     ms.Write(defaultLangSysTable, 0, defaultLangSysTable.Length);
03525 
03526                     for (int i = 0; i < langSysTables[i].Length; i++)
03527                     {
03528                         ms.Write(langSysTables[i], 0, langSysTables[i].Length);
03529                     }
03530 
03531                     return ms.ToArray();
03532                 }
03533             }
03534         }
03535 
03536 
03537         internal class ScriptList
03538         {
03539             public ScriptTable[] ScriptRecords;
03540             public ScriptList(Stream fs)
03541             {
03542                 long startPosition = fs.Position;
03543 
03544                 ushort scriptCount = fs.ReadUShort();
03545 
03546                 (string, ushort)[] scriptRecords = new (string, ushort)[scriptCount];
03547 
03548                 for (int i = 0; i < scriptCount; i++)
03549                 {
03550                     StringBuilder scriptTag = new StringBuilder(4);
03551                     scriptTag.Append((char)fs.ReadByte());
03552                     scriptTag.Append((char)fs.ReadByte());
03553                     scriptTag.Append((char)fs.ReadByte());
03554                     scriptTag.Append((char)fs.ReadByte());
03555 
```

```
03556                   scriptRecords[i] = (scriptTag.ToString(), fs.ReadUShort());
03557               }
03558
03559               this.ScriptRecords = new ScriptTable[scriptCount];
03560
03561               for (int i = 0; i < scriptCount; i++)
03562               {
03563                   fs.Seek(startPosition + scriptRecords[i].Item2, SeekOrigin.Begin);
03564                   this.ScriptRecords[i] = new ScriptTable(scriptRecords[i].Item1, fs);
03565               }
03566           }
03567
03568           public byte[] GetBytes()
03569           {
03570               using (MemoryStream ms = new MemoryStream())
03571               {
03572                   int offset = 2 + 6 * this.ScriptRecords.Length;
03573
03574                   ms.WriteUShort((ushort)this.ScriptRecords.Length);
03575
03576                   byte[][] scripts = new byte[this.ScriptRecords.Length][];
03577                   int[] offsets = new int[this.ScriptRecords.Length];
03578
03579                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03580                   {
03581                       offsets[i] = offset;
03582                       scripts[i] = this.ScriptRecords[i].GetBytes();
03583                       offset += scripts[i].Length;
03584                   }
03585
03586                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03587                   {
03588                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[0]);
03589                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[1]);
03590                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[2]);
03591                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[3]);
03592
03593                       ms.WriteUShort((ushort)offsets[i]);
03594                   }
03595
03596                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03597                   {
03598                       ms.Write(scripts[i], 0, scripts[i].Length);
03599                   }
03600
03601                   return ms.ToArray();
03602               }
03603           }
03604       }
03605
03606       internal class LookupTable
03607       {
03608           public ushort LookupType;
03609           public ushort LookupFlag;
03610           public ITrueTypeTable[] SubTables;
03611           public ushort MarkFilteringSet;
03612
03613           public LookupTable(Stream fs)
03614           {
03615               long startPosition = fs.Position;
03616
03617               this.LookupType = fs.ReadUShort();
03618               this.LookupFlag = fs.ReadUShort();
03619
03620               ushort subTableCount = fs.ReadUShort();
03621               ushort[] subTableOffsets = new ushort[subTableCount];
03622
03623               for (int i = 0; i < subTableCount; i++)
03624               {
03625                   subTableOffsets[i] = fs.ReadUShort();
03626               }
03627
03628               if ((this.LookupFlag & 0x0010) != 0)
03629               {
03630                   this.MarkFilteringSet = fs.ReadUShort();
03631               }
03632
03633               if (this.LookupType == 1)
03634               {
03635                   this.SubTables = new ITrueTypeTable[subTableCount];
03636
03637                   for (int i = 0; i < subTableCount; i++)
03638                   {
03639                       fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03640                       this.SubTables[i] = new SinglePosSubtable(fs);
03641                   }
03642               }
```

```
03643                 else if (this.LookupType == 2)
03644                 {
03645                     this.SubTables = new ITrueTypeTable[subTableCount];
03646 
03647                     for (int i = 0; i < subTableCount; i++)
03648                     {
03649                         fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03650                         this.SubTables[i] = new PairPosSubtable(fs);
03651                     }
03652                 }
03653                 else if (this.LookupType == 9)
03654                 {
03655                     for (int i = 0; i < subTableCount; i++)
03656                     {
03657                         fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03658 
03659                         long extensionSubtableStartPosition = fs.Position;
03660 
03661                         ushort posFormat = fs.ReadUShort(); // 1
03662                         ushort extensionLookupType = fs.ReadUShort();
03663                         uint offset = fs.ReadUInt();
03664 
03665                         if (i == 0)
03666                         {
03667                             this.LookupType = extensionLookupType;
03668 
03669                             if (this.LookupType == 1 || this.LookupType == 2)
03670                             {
03671                                 this.SubTables = new ITrueTypeTable[subTableCount];
03672                             }
03673                             else
03674                             {
03675                                 this.SubTables = new ITrueTypeTable[0];
03676                             }
03677                         }
03678 
03679                         if (this.LookupType == 1)
03680                         {
03681                             fs.Seek(extensionSubtableStartPosition + offset, SeekOrigin.Begin);
03682                             this.SubTables[i] = new SinglePosSubtable(fs);
03683                         }
03684                         else if (this.LookupType == 2)
03685                         {
03686                             fs.Seek(extensionSubtableStartPosition + offset, SeekOrigin.Begin);
03687                             this.SubTables[i] = new PairPosSubtable(fs);
03688                         }
03689                     }
03690                 }
03691                 else
03692                 {
03693                     this.SubTables = new ITrueTypeTable[0];
03694                 }
03695             }
03696 
03697             public byte[] GetBytes()
03698             {
03699                 using (MemoryStream ms = new MemoryStream())
03700                 {
03701                     ms.WriteUShort(this.LookupType);
03702                     ms.WriteUShort(this.LookupFlag);
03703                     ms.WriteUShort((ushort)this.SubTables.Length);
03704 
03705                     int offset = 6 + 2 * this.SubTables.Length;
03706 
03707                     if ((this.LookupFlag & 0x0010) != 0)
03708                     {
03709                         offset += 2;
03710                     }
03711 
03712                     byte[][] subTables = new byte[this.SubTables.Length][];
03713                     int[] offsets = new int[this.SubTables.Length];
03714 
03715                     for (int i = 0; i < this.SubTables.Length; i++)
03716                     {
03717                         offsets[i] = offset;
03718                         subTables[i] = this.SubTables[i].GetBytes();
03719                         offset += subTables[i].Length;
03720                     }
03721 
03722                     for (int i = 0; i < this.SubTables.Length; i++)
03723                     {
03724                         ms.WriteUShort((ushort)offsets[i]);
03725                     }
03726 
03727                     if ((this.LookupFlag & 0x0010) != 0)
03728                     {
03729                         ms.WriteUShort(this.MarkFilteringSet);
```

```
03730                     }
03731
03732                     for (int i = 0; i < this.SubTables.Length; i++)
03733                     {
03734                         ms.Write(subTables[i], 0, subTables[i].Length);
03735                     }
03736
03737                     return ms.ToArray();
03738                 }
03739             }
03740         }
03741
03742         internal class LookupList
03743         {
03744             public LookupTable[] LookupTables;
03745
03746             public LookupList(Stream fs)
03747             {
03748                 long startPosition = fs.Position;
03749
03750                 ushort lookupCount = fs.ReadUShort();
03751                 ushort[] lookupOffsets = new ushort[lookupCount];
03752
03753                 for (int i = 0; i < lookupCount; i++)
03754                 {
03755                     lookupOffsets[i] = fs.ReadUShort();
03756                 }
03757
03758                 this.LookupTables = new LookupTable[lookupCount];
03759
03760                 for (int i = 0; i < lookupCount; i++)
03761                 {
03762                     fs.Seek(startPosition + lookupOffsets[i], SeekOrigin.Begin);
03763                     this.LookupTables[i] = new LookupTable(fs);
03764                 }
03765             }
03766
03767             public byte[] GetBytes()
03768             {
03769                 using (MemoryStream ms = new MemoryStream())
03770                 {
03771                     ms.WriteUShort((ushort)this.LookupTables.Length);
03772
03773                     int offset = 2 + 2 * this.LookupTables.Length;
03774
03775                     int[] offsets = new int[this.LookupTables.Length];
03776                     byte[][] lookupTables = new byte[this.LookupTables.Length][];
03777
03778                     for (int i = 0; i < this.LookupTables.Length; i++)
03779                     {
03780                         offsets[i] = offset;
03781                         lookupTables[i] = this.LookupTables[i].GetBytes();
03782                         offset += lookupTables[i].Length;
03783                     }
03784
03785                     for (int i = 0; i < this.LookupTables.Length; i++)
03786                     {
03787                         ms.WriteUShort((ushort)offsets[i]);
03788                     }
03789
03790                     for (int i = 0; i < this.LookupTables.Length; i++)
03791                     {
03792                         ms.Write(lookupTables[i], 0, lookupTables[i].Length);
03793                     }
03794
03795                     return ms.ToArray();
03796                 }
03797             }
03798         }
03799
03800         internal class CoverageTable
03801         {
03802             public struct RangeRecord
03803             {
03804                 public ushort StartGlyphID;
03805                 public ushort EndGlyphID;
03806                 public ushort StartCoverageIndex;
03807
03808                 public RangeRecord(Stream fs)
03809                 {
03810                     this.StartGlyphID = fs.ReadUShort();
03811                     this.EndGlyphID = fs.ReadUShort();
03812                     this.StartCoverageIndex = fs.ReadUShort();
03813                 }
03814             }
03815
03816             public ushort CoverageFormat;
```

```
03817
03818             public ushort[] GlyphArray;
03819             public RangeRecord[] RangeRecords;
03820
03821             public int ContainsGlyph(int glyphIndex)
03822             {
03823                 if (this.CoverageFormat == 1)
03824                 {
03825                     for (int i = 0; i < this.GlyphArray.Length; i++)
03826                     {
03827                         if (this.GlyphArray[i] == glyphIndex)
03828                         {
03829                             return i;
03830                         }
03831                     }
03832
03833                     return -1;
03834                 }
03835                 else if (this.CoverageFormat == 2)
03836                 {
03837                     for (int i = 0; i < this.RangeRecords.Length; i++)
03838                     {
03839                         if (this.RangeRecords[i].StartGlyphID <= glyphIndex &&
      this.RangeRecords[i].EndGlyphID >= glyphIndex)
03840                         {
03841                             return this.RangeRecords[i].StartCoverageIndex + (glyphIndex -
      this.RangeRecords[i].StartGlyphID);
03842                         }
03843                     }
03844
03845                     return -1;
03846                 }
03847                 else
03848                 {
03849                     return -1;
03850                 }
03851             }
03852
03853             public CoverageTable(Stream fs)
03854             {
03855                 this.CoverageFormat = fs.ReadUShort();
03856
03857                 if (this.CoverageFormat == 1)
03858                 {
03859                     ushort glyphCount = fs.ReadUShort();
03860                     this.GlyphArray = new ushort[glyphCount];
03861                     for (int i = 0; i < glyphCount; i++)
03862                     {
03863                         this.GlyphArray[i] = fs.ReadUShort();
03864                     }
03865                 }
03866                 else if (this.CoverageFormat == 2)
03867                 {
03868                     ushort rangeCount = fs.ReadUShort();
03869
03870                     this.RangeRecords = new RangeRecord[rangeCount];
03871
03872                     for (int i = 0; i < rangeCount; i++)
03873                     {
03874                         this.RangeRecords[i] = new RangeRecord(fs);
03875                     }
03876                 }
03877             }
03878
03879             public byte[] GetBytes()
03880             {
03881                 if (this.CoverageFormat == 1)
03882                 {
03883                     using (MemoryStream ms = new MemoryStream(4 + 2 * this.GlyphArray.Length))
03884                     {
03885                         ms.WriteUShort(this.CoverageFormat);
03886                         ms.WriteUShort((ushort)this.GlyphArray.Length);
03887                         for (int i = 0; i < this.GlyphArray.Length; i++)
03888                         {
03889                             ms.WriteUShort(this.GlyphArray[i]);
03890                         }
03891
03892                         return ms.ToArray();
03893                     }
03894                 }
03895                 else if (this.CoverageFormat == 2)
03896                 {
03897                     using (MemoryStream ms = new MemoryStream(4 + 6 * this.RangeRecords.Length))
03898                     {
03899                         ms.WriteUShort(this.CoverageFormat);
03900                         ms.WriteUShort((ushort)this.RangeRecords.Length);
03901                         for (int i = 0; i < this.RangeRecords.Length; i++)
```

```
03902                         {
03903                             ms.WriteUShort(this.RangeRecords[i].StartGlyphID);
03904                             ms.WriteUShort(this.RangeRecords[i].EndGlyphID);
03905                             ms.WriteUShort(this.RangeRecords[i].StartCoverageIndex);
03906                         }
03907 
03908                         return ms.ToArray();
03909                     }
03910                 }
03911                 else
03912                 {
03913                     using (MemoryStream ms = new MemoryStream(4 + 2 * this.GlyphArray.Length))
03914                     {
03915                         ms.WriteUShort(this.CoverageFormat);
03916 
03917                         return ms.ToArray();
03918                     }
03919                 }
03920             }
03921         }
03922 
03923         internal class ClassDefinitionTable
03924         {
03925             public struct ClassRangeRecord
03926             {
03927                 public ushort StartGlyphID;
03928                 public ushort EndGlyphID;
03929                 public ushort Class;
03930 
03931                 public ClassRangeRecord(Stream fs)
03932                 {
03933                     this.StartGlyphID = fs.ReadUShort();
03934                     this.EndGlyphID = fs.ReadUShort();
03935                     this.Class = fs.ReadUShort();
03936                 }
03937             }
03938 
03939             public ushort ClassFormat;
03940             public ushort StartGlyphID;
03941             public ushort[] ClassValueArray;
03942 
03943             public ClassRangeRecord[] ClassRangeRecords;
03944 
03945             public ClassDefinitionTable(Stream fs)
03946             {
03947                 this.ClassFormat = fs.ReadUShort();
03948 
03949                 if (this.ClassFormat == 1)
03950                 {
03951                     this.StartGlyphID = fs.ReadUShort();
03952                     ushort glyphCount = fs.ReadUShort();
03953 
03954                     this.ClassValueArray = new ushort[glyphCount];
03955 
03956                     for (int i = 0; i < this.ClassValueArray.Length; i++)
03957                     {
03958                         this.ClassValueArray[i] = fs.ReadUShort();
03959                     }
03960                 }
03961                 else if (this.ClassFormat == 2)
03962                 {
03963                     ushort classRangeCount = fs.ReadUShort();
03964 
03965                     this.ClassRangeRecords = new ClassRangeRecord[classRangeCount];
03966 
03967                     for (int i = 0; i < classRangeCount; i++)
03968                     {
03969                         this.ClassRangeRecords[i] = new ClassRangeRecord(fs);
03970                     }
03971                 }
03972             }
03973 
03974             public int GetClass(int glyphIndex)
03975             {
03976                 if (this.ClassFormat == 1)
03977                 {
03978                     if (glyphIndex >= StartGlyphID && glyphIndex - StartGlyphID <
    this.ClassValueArray.Length)
03979                     {
03980                         return this.ClassValueArray[glyphIndex - StartGlyphID];
03981                     }
03982                     else
03983                     {
03984                         return -1;
03985                     }
03986                 }
03987                 else if (this.ClassFormat == 2)
```

```
03988                 {
03989                     for (int i = 0; i < this.ClassRangeRecords.Length; i++)
03990                     {
03991                         if (this.ClassRangeRecords[i].StartGlyphID <= glyphIndex &&
      this.ClassRangeRecords[i].EndGlyphID >= glyphIndex)
03992                         {
03993                             return this.ClassRangeRecords[i].Class;
03994                         }
03995                     }
03996                     return -1;
03997                 }
03998                 else
03999                 {
04000                     return -1;
04001                 }
04002             }
04003 
04004             public byte[] GetBytes()
04005             {
04006                 if (this.ClassFormat == 1)
04007                 {
04008                     using (MemoryStream ms = new MemoryStream(6 + 2 * this.ClassValueArray.Length))
04009                     {
04010                         ms.WriteUShort(this.ClassFormat);
04011                         ms.WriteUShort(this.StartGlyphID);
04012                         ms.WriteUShort((ushort)this.ClassValueArray.Length);
04013 
04014                         for (int i = 0; i < this.ClassValueArray.Length; i++)
04015                         {
04016                             ms.WriteUShort(this.ClassValueArray[i]);
04017                         }
04018 
04019                         return ms.ToArray();
04020                     }
04021                 }
04022                 else if (this.ClassFormat == 2)
04023                 {
04024                     using (MemoryStream ms = new MemoryStream(4 + 6 * this.ClassRangeRecords.Length))
04025                     {
04026                         ms.WriteUShort(this.ClassFormat);
04027                         ms.WriteUShort((ushort)this.ClassRangeRecords.Length);
04028 
04029                         for (int i = 0; i < this.ClassRangeRecords.Length; i++)
04030                         {
04031                             ms.WriteUShort(this.ClassRangeRecords[i].StartGlyphID);
04032                             ms.WriteUShort(this.ClassRangeRecords[i].EndGlyphID);
04033                             ms.WriteUShort(this.ClassRangeRecords[i].Class);
04034                         }
04035 
04036                         return ms.ToArray();
04037                     }
04038                 }
04039                 else
04040                 {
04041                     using (MemoryStream ms = new MemoryStream(2))
04042                     {
04043                         ms.WriteUShort(this.ClassFormat);
04044 
04045                         return ms.ToArray();
04046                     }
04047                 }
04048             }
04049         }
04050 
04051         internal class FeatureTable
04052         {
04053             public string Tag;
04054             public byte[] FeatureParams;
04055             public ushort[] LookupListIndices;
04056 
04057             public FeatureTable(string tag, Stream fs)
04058             {
04059                 this.Tag = tag;
04060 
04061                 long startPosition = fs.Position;
04062 
04063                 ushort featureParamsOffset = fs.ReadUShort();
04064 
04065                 ushort lookupIndexCount = fs.ReadUShort();
04066                 this.LookupListIndices = new ushort[lookupIndexCount];
04067 
04068                 for (int i = 0; i < lookupIndexCount; i++)
04069                 {
04070                     this.LookupListIndices[i] = fs.ReadUShort();
04071                 }
04072 
04073                 if (featureParamsOffset != 0 && tag == "size")
```

```
04074                 {
04075                     fs.Seek(startPosition + featureParamsOffset, SeekOrigin.Begin);
04076 
04077                     this.FeatureParams = new byte[10];
04078 
04079                     for (int i = 0; i < 10; i++)
04080                     {
04081                         this.FeatureParams[i] = (byte)fs.ReadByte();
04082                     }
04083                 }
04084                 else
04085                 {
04086                     this.FeatureParams = null;
04087                 }
04088             }
04089 
04090             public byte[] GetBytes()
04091             {
04092                 using (MemoryStream ms = new MemoryStream())
04093                 {
04094                     if (this.FeatureParams != null)
04095                     {
04096                         ms.WriteUShort((ushort)(4 + 2 * this.LookupListIndices.Length));
04097                     }
04098                     else
04099                     {
04100                         ms.WriteUShort(0);
04101                     }
04102 
04103                     ms.WriteUShort((ushort)this.LookupListIndices.Length);
04104 
04105                     for (int i = 0; i < this.LookupListIndices.Length; i++)
04106                     {
04107                         ms.WriteUShort(this.LookupListIndices[i]);
04108                     }
04109 
04110                     if (this.FeatureParams != null)
04111                     {
04112                         ms.Write(this.FeatureParams, 0, this.FeatureParams.Length);
04113                     }
04114 
04115                     return ms.ToArray();
04116                 }
04117             }
04118         }
04119 
04120         internal class FeatureList
04121         {
04122             public FeatureTable[] FeatureRecords;
04123 
04124             public FeatureList(Stream fs)
04125             {
04126                 long startPosition = fs.Position;
04127 
04128                 ushort featureCount = fs.ReadUShort();
04129 
04130                 (string, ushort)[] featureRecords = new (string, ushort)[featureCount];
04131 
04132                 for (int i = 0; i < featureCount; i++)
04133                 {
04134                     StringBuilder featureTag = new StringBuilder(4);
04135                     featureTag.Append((char)fs.ReadByte());
04136                     featureTag.Append((char)fs.ReadByte());
04137                     featureTag.Append((char)fs.ReadByte());
04138                     featureTag.Append((char)fs.ReadByte());
04139 
04140                     featureRecords[i] = (featureTag.ToString(), fs.ReadUShort());
04141                 }
04142 
04143                 this.FeatureRecords = new FeatureTable[featureCount];
04144 
04145                 for (int i = 0; i < featureCount; i++)
04146                 {
04147                     fs.Seek(startPosition + featureRecords[i].Item2, SeekOrigin.Begin);
04148 
04149                     this.FeatureRecords[i] = new FeatureTable(featureRecords[i].Item1, fs);
04150                 }
04151             }
04152 
04153             public byte[] GetBytes()
04154             {
04155                 using (MemoryStream ms = new MemoryStream())
04156                 {
04157                     int offset = 2 + 6 * this.FeatureRecords.Length;
04158 
04159                     ms.WriteUShort((ushort)this.FeatureRecords.Length);
04160 
```

```
04161                   byte[][] featureRecords = new byte[this.FeatureRecords.Length][];
04162                   int[] offsets = new int[this.FeatureRecords.Length];
04163
04164                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04165                   {
04166                       offsets[i] = offset;
04167                       featureRecords[i] = this.FeatureRecords[i].GetBytes();
04168                       offset += featureRecords[i].Length;
04169                   }
04170
04171                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04172                   {
04173                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[0]);
04174                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[1]);
04175                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[2]);
04176                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[3]);
04177
04178                       ms.WriteUShort((ushort)offsets[i]);
04179                   }
04180
04181                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04182                   {
04183                       ms.Write(featureRecords[i], 0, featureRecords[i].Length);
04184                   }
04185
04186                   return ms.ToArray();
04187               }
04188           }
04189       }
04190
04191       internal class ValueRecord
04192       {
04193           public short XPlacement;
04194           public short YPlacement;
04195           public short XAdvance;
04196           public short YAdvance;
04197           public ushort ValueFormat;
04198
04199           public ValueRecord(Stream fs, ushort valueFormat)
04200           {
04201               this.ValueFormat = valueFormat;
04202
04203               if ((valueFormat & 0x0001) != 0)
04204               {
04205                   this.XPlacement = fs.ReadShort();
04206               }
04207
04208               if ((valueFormat & 0x0002) != 0)
04209               {
04210                   this.YPlacement = fs.ReadShort();
04211               }
04212
04213               if ((valueFormat & 0x0004) != 0)
04214               {
04215                   this.XAdvance = fs.ReadShort();
04216               }
04217
04218               if ((valueFormat & 0x0008) != 0)
04219               {
04220                   this.YAdvance = fs.ReadShort();
04221               }
04222
04223               if ((valueFormat & 0x0010) != 0)
04224               {
04225                   fs.ReadUShort();
04226               }
04227
04228               if ((valueFormat & 0x0020) != 0)
04229               {
04230                   fs.ReadUShort();
04231               }
04232
04233               if ((valueFormat & 0x0040) != 0)
04234               {
04235                   fs.ReadUShort();
04236               }
04237
04238               if ((valueFormat & 0x0080) != 0)
04239               {
04240                   fs.ReadUShort();
04241               }
04242           }
04243
04244           public byte[] GetBytes()
04245           {
04246               using (MemoryStream ms = new MemoryStream())
04247               {
```

```
04248                   if ((this.ValueFormat & 0x0001) != 0)
04249                   {
04250                       ms.WriteShort(this.XPlacement);
04251                   }
04252
04253                   if ((this.ValueFormat & 0x0002) != 0)
04254                   {
04255                       ms.WriteShort(this.YPlacement);
04256                   }
04257
04258                   if ((this.ValueFormat & 0x0004) != 0)
04259                   {
04260                       ms.WriteShort(this.XAdvance);
04261                   }
04262
04263                   if ((this.ValueFormat & 0x0008) != 0)
04264                   {
04265                       ms.WriteShort(this.YAdvance);
04266                   }
04267
04268                   if ((this.ValueFormat & 0x0010) != 0)
04269                   {
04270                       ms.WriteUShort(0);
04271                   }
04272
04273                   if ((this.ValueFormat & 0x0020) != 0)
04274                   {
04275                       ms.WriteUShort(0);
04276                   }
04277
04278                   if ((this.ValueFormat & 0x0040) != 0)
04279                   {
04280                       ms.WriteUShort(0);
04281                   }
04282
04283                   if ((this.ValueFormat & 0x0080) != 0)
04284                   {
04285                       ms.WriteUShort(0);
04286                   }
04287
04288                   return ms.ToArray();
04289               }
04290           }
04291       }
04292
04293       internal class SinglePosSubtable :  ITrueTypeTable
04294       {
04295           public ushort PosFormat;
04296           public CoverageTable CoverageTable;
04297           public ushort ValueFormat;
04298           public ValueRecord ValueRecord;
04299           public ValueRecord[] ValueRecords;
04300
04301           public SinglePosSubtable(Stream fs)
04302           {
04303               long startPosition = fs.Position;
04304
04305               this.PosFormat = fs.ReadUShort();
04306
04307               if (this.PosFormat == 1)
04308               {
04309                   ushort coverageOffset = fs.ReadUShort();
04310                   this.ValueFormat = fs.ReadUShort();
04311                   this.ValueRecord = new ValueRecord(fs, this.ValueFormat);
04312
04313                   fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04314                   this.CoverageTable = new CoverageTable(fs);
04315               }
04316               else if (this.PosFormat == 2)
04317               {
04318                   ushort coverageOffset = fs.ReadUShort();
04319                   this.ValueFormat = fs.ReadUShort();
04320
04321                   ushort valueCount = fs.ReadUShort();
04322
04323                   this.ValueRecords = new ValueRecord[valueCount];
04324
04325                   for (int i = 0; i < valueCount; i++)
04326                   {
04327                       this.ValueRecords[i] = new ValueRecord(fs, this.ValueFormat);
04328                   }
04329
04330                   fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04331                   this.CoverageTable = new CoverageTable(fs);
04332               }
04333           }
04334
```

```
04335             public byte[] GetBytes()
04336             {
04337                 using (MemoryStream ms = new MemoryStream())
04338                 {
04339                     ms.WriteUShort(this.PosFormat);
04340 
04341                     if (this.PosFormat == 1)
04342                     {
04343                         int offset = 22;
04344 
04345                         ms.WriteUShort((ushort)offset);
04346                         ms.WriteUShort(this.ValueFormat);
04347 
04348                         byte[] valueBytes = this.ValueRecord.GetBytes();
04349 
04350                         ms.Write(valueBytes, 0, valueBytes.Length);
04351 
04352                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04353 
04354                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04355                     }
04356                     else if (this.PosFormat == 2)
04357                     {
04358                         int offset = 8 + 16 * ValueRecords.Length;
04359 
04360                         ms.WriteUShort((ushort)offset);
04361                         ms.WriteUShort(this.ValueFormat);
04362                         ms.WriteUShort((ushort)this.ValueRecords.Length);
04363 
04364                         for (int i = 0; i < this.ValueRecords.Length; i++)
04365                         {
04366                             byte[] valueBytes = this.ValueRecords[i].GetBytes();
04367 
04368                             ms.Write(valueBytes, 0, valueBytes.Length);
04369                         }
04370 
04371                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04372 
04373                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04374                     }
04375 
04376                     return ms.ToArray();
04377                 }
04378             }
04379         }
04380 
04381         internal class PairValueRecord
04382         {
04383             public ushort SecondGlyph;
04384             public ValueRecord ValueRecord1;
04385             public ValueRecord ValueRecord2;
04386 
04387             public PairValueRecord(Stream fs, ushort valueFormat1, ushort valueFormat2)
04388             {
04389                 this.SecondGlyph = fs.ReadUShort();
04390                 this.ValueRecord1 = new ValueRecord(fs, valueFormat1);
04391                 this.ValueRecord2 = new ValueRecord(fs, valueFormat2);
04392             }
04393 
04394             public byte[] GetBytes()
04395             {
04396                 using (MemoryStream ms = new MemoryStream())
04397                 {
04398                     ms.WriteUShort(SecondGlyph);
04399 
04400                     byte[] bytes = this.ValueRecord1.GetBytes();
04401                     ms.Write(bytes, 0, bytes.Length);
04402 
04403                     bytes = this.ValueRecord2.GetBytes();
04404                     ms.Write(bytes, 0, bytes.Length);
04405 
04406                     return ms.ToArray();
04407                 }
04408             }
04409         }
04410 
04411         internal class Class2Record
04412         {
04413             public ValueRecord ValueRecord1;
04414             public ValueRecord ValueRecord2;
04415 
04416             public Class2Record(Stream fs, ushort valueFormat1, ushort valueFormat2)
04417             {
04418                 this.ValueRecord1 = new ValueRecord(fs, valueFormat1);
04419                 this.ValueRecord2 = new ValueRecord(fs, valueFormat2);
04420             }
04421 
```

```
04422             public byte[] GetBytes()
04423             {
04424                 using (MemoryStream ms = new MemoryStream())
04425                 {
04426                     byte[] bytes = this.ValueRecord1.GetBytes();
04427                     ms.Write(bytes, 0, bytes.Length);
04428 
04429                     bytes = this.ValueRecord2.GetBytes();
04430                     ms.Write(bytes, 0, bytes.Length);
04431 
04432                     return ms.ToArray();
04433                 }
04434             }
04435         }
04436 
04437 /// <summary>
04438 /// Contains information describing how the position of two glyphs in a kerning pair should be
      altered.
04439 /// </summary>
04440         public class PairKerning
04441         {
04442 /// <summary>
04443 /// This vector contains the displacement that should be applied to the first glyph of the pair.
04444 /// </summary>
04445             public Point Glyph1Placement { get; }
04446 
04447 /// <summary>
04448 /// This vector describes how the advance width of the first glyph should be altered.
04449 /// </summary>
04450             public Point Glyph1Advance { get; }
04451 
04452 /// <summary>
04453 /// This vector contains the displacement that should be applied to the second glyph of the pair.
04454 /// </summary>
04455             public Point Glyph2Placement { get; }
04456 
04457 /// <summary>
04458 /// This vector describes how the advance width of the second glyph should be altered.
04459 /// </summary>
04460             public Point Glyph2Advance { get; }
04461 
04462             internal PairKerning(Point glyph1Placement, Point glyph1Advance, Point glyph2Placement,
      Point glyph2Advance)
04463             {
04464                 this.Glyph1Placement = glyph1Placement;
04465                 this.Glyph1Advance = glyph1Advance;
04466                 this.Glyph2Placement = glyph2Placement;
04467                 this.Glyph2Advance = glyph2Advance;
04468             }
04469         }
04470 
04471         internal class PairPosSubtable :  ITrueTypeTable
04472         {
04473             public ushort PosFormat;
04474             public CoverageTable CoverageTable;
04475 
04476             public ushort ValueFormat1;
04477             public ushort ValueFormat2;
04478 
04479             public PairValueRecord[][] PairSets;
04480 
04481             public ClassDefinitionTable ClassDef1;
04482             public ClassDefinitionTable ClassDef2;
04483             public Class2Record[][] Class1Records;
04484 
04485             public PairPosSubtable(Stream fs)
04486             {
04487                 long startPosition = fs.Position;
04488 
04489                 this.PosFormat = fs.ReadUShort();
04490 
04491                 if (this.PosFormat == 1)
04492                 {
04493                     ushort coverageOffset = fs.ReadUShort();
04494 
04495                     this.ValueFormat1 = fs.ReadUShort();
04496                     this.ValueFormat2 = fs.ReadUShort();
04497 
04498                     ushort pairSetCount = fs.ReadUShort();
04499 
04500                     ushort[] pairSetOffsets = new ushort[pairSetCount];
04501 
04502                     for (int i = 0; i < pairSetCount; i++)
04503                     {
04504                         pairSetOffsets[i] = fs.ReadUShort();
04505                     }
04506 
```

```
04507                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04508                     this.CoverageTable = new CoverageTable(fs);
04509 
04510                     this.PairSets = new PairValueRecord[pairSetCount][];
04511 
04512                     for (int i = 0; i < pairSetCount; i++)
04513                     {
04514                         fs.Seek(startPosition + pairSetOffsets[i], SeekOrigin.Begin);
04515 
04516                         ushort pairValueCount = fs.ReadUShort();
04517 
04518                         this.PairSets[i] = new PairValueRecord[pairValueCount];
04519 
04520                         for (int j = 0; j < pairValueCount; j++)
04521                         {
04522                             this.PairSets[i][j] = new PairValueRecord(fs, this.ValueFormat1,
      this.ValueFormat2);
04523                         }
04524                     }
04525                 }
04526                 else if (this.PosFormat == 2)
04527                 {
04528                     ushort coverageOffset = fs.ReadUShort();
04529 
04530                     this.ValueFormat1 = fs.ReadUShort();
04531                     this.ValueFormat2 = fs.ReadUShort();
04532 
04533                     ushort class1DefOffset = fs.ReadUShort();
04534                     ushort class2DefOffset = fs.ReadUShort();
04535 
04536                     ushort class1Count = fs.ReadUShort();
04537                     ushort class2Count = fs.ReadUShort();
04538 
04539                     this.Class1Records = new Class2Record[class1Count][];
04540 
04541                     for (int i = 0; i < class1Count; i++)
04542                     {
04543                         this.Class1Records[i] = new Class2Record[class2Count];
04544 
04545                         for (int j = 0; j < class2Count; j++)
04546                         {
04547                             this.Class1Records[i][j] = new Class2Record(fs, this.ValueFormat1,
      this.ValueFormat2);
04548                         }
04549                     }
04550 
04551                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04552                     this.CoverageTable = new CoverageTable(fs);
04553 
04554                     fs.Seek(startPosition + class1DefOffset, SeekOrigin.Begin);
04555                     this.ClassDef1 = new ClassDefinitionTable(fs);
04556 
04557                     fs.Seek(startPosition + class2DefOffset, SeekOrigin.Begin);
04558 
04559                     this.ClassDef2 = new ClassDefinitionTable(fs);
04560                 }
04561             }
04562 
04563             public PairKerning GetKerning(int glyph1CoverageIndex, int glyph1Index, int glyph2Index)
04564             {
04565                 if (this.PosFormat == 1)
04566                 {
04567                     for (int i = 0; i < this.PairSets[glyph1CoverageIndex].Length; i++)
04568                     {
04569                         if (this.PairSets[glyph1CoverageIndex][i].SecondGlyph == glyph2Index)
04570                         {
04571                             return new PairKerning(new
      Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord1.XPlacement,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord1.YPlacement),
04572                                 new Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord1.XAdvance,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord1.YAdvance),
04573                                 new
      Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord2.XPlacement,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord2.YPlacement),
04574                                 new Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord2.XAdvance,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord2.YAdvance));
04575                         }
04576                     }
04577 
04578                     return null;
04579                 }
04580                 else if (this.PosFormat == 2)
04581                 {
04582                     int glyph1Class = this.ClassDef1.GetClass(glyph1Index);
04583                     int glyph2Class = this.ClassDef2.GetClass(glyph2Index);
04584 
04585                     if (glyph1Class < 0 || glyph2Class < 0)
```

```
04586                     {
04587                         return null;
04588                     }
04589 
04590                     return new PairKerning(new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.XPlacement,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.YPlacement),
04591                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.XAdvance,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.YAdvance),
04592                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.XPlacement,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.YPlacement),
04593                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.XAdvance,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.YAdvance));
04594                 }
04595                 else
04596                 {
04597                     return null;
04598                 }
04599             }
04600 
04601             public byte[] GetBytes()
04602             {
04603                 using (MemoryStream ms = new MemoryStream())
04604                 {
04605                     ms.WriteUShort(this.PosFormat);
04606 
04607                     if (this.PosFormat == 1)
04608                     {
04609                         int offset = 10 + 2 * PairSets.Length;
04610 
04611                         ms.WriteUShort((ushort)offset);
04612 
04613                         ms.WriteUShort((ushort)ValueFormat1);
04614                         ms.WriteUShort((ushort)ValueFormat2);
04615 
04616                         ms.WriteUShort((ushort)this.PairSets.Length);
04617 
04618                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04619 
04620                         offset += coverageBytes.Length;
04621 
04622                         byte[][][] pairSets = new byte[this.PairSets.Length][][];
04623                         int[] offsets = new int[this.PairSets.Length];
04624 
04625                         for (int i = 0; i < this.PairSets.Length; i++)
04626                         {
04627                             offsets[i] = offset;
04628 
04629                             pairSets[i] = new byte[this.PairSets[i].Length][];
04630 
04631                             for (int j = 0; j < this.PairSets[i].Length; j++)
04632                             {
04633                                 pairSets[i][j] = this.PairSets[i][j].GetBytes();
04634                                 offset += pairSets[i][j].Length;
04635                             }
04636                         }
04637 
04638                         for (int i = 0; i < offsets.Length; i++)
04639                         {
04640                             ms.WriteUShort((ushort)offsets[i]);
04641                         }
04642 
04643                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04644 
04645                         for (int i = 0; i < pairSets.Length; i++)
04646                         {
04647                             for (int j = 0; j < pairSets[i].Length; j++)
04648                             {
04649                                 ms.Write(pairSets[i][j], 0, pairSets[i][j].Length);
04650                             }
04651                         }
04652 
04653                         return ms.ToArray();
04654                     }
04655                     else if (this.PosFormat == 2)
04656                     {
04657                         int offset = 16;
04658 
04659                         byte[][][] class1Records = new byte[this.Class1Records.Length][][];
04660 
04661                         for (int i = 0; i < class1Records.Length; i++)
04662                         {
04663                             class1Records[i] = new byte[this.Class1Records[i].Length][];
04664 
```

```
04665                           for (int j = 0; j < class1Records[i].Length; j++)
04666                           {
04667                               class1Records[i][j] = this.Class1Records[i][j].GetBytes();
04668                               offset += class1Records[i][j].Length;
04669                           }
04670                       }
04671
04672                       ms.WriteUShort((ushort)offset);
04673
04674                       ms.WriteUShort(this.ValueFormat1);
04675                       ms.WriteUShort(this.ValueFormat2);
04676
04677                       byte[] coverageBytes = this.CoverageTable.GetBytes();
04678
04679                       offset += coverageBytes.Length;
04680
04681                       ms.WriteUShort((ushort)offset);
04682
04683                       byte[] classDef1Bytes = this.ClassDef1.GetBytes();
04684
04685                       offset += classDef1Bytes.Length;
04686
04687                       ms.WriteUShort((ushort)offset);
04688
04689                       ms.WriteUShort((ushort)this.Class1Records.Length);
04690                       ms.WriteUShort((ushort)this.Class1Records[0].Length);
04691
04692                       for (int i = 0; i < class1Records.Length; i++)
04693                       {
04694                           for (int j = 0; j < class1Records[i].Length; j++)
04695                           {
04696                               ms.Write(class1Records[i][j], 0, class1Records[i][j].Length);
04697                           }
04698                       }
04699
04700                       ms.Write(coverageBytes, 0, coverageBytes.Length);
04701                       ms.Write(classDef1Bytes, 0, classDef1Bytes.Length);
04702
04703                       byte[] classDef2Bytes = this.ClassDef2.GetBytes();
04704                       ms.Write(classDef2Bytes, 0, classDef2Bytes.Length);
04705
04706                       return ms.ToArray();
04707                   }
04708                   else
04709                   {
04710                       return ms.ToArray();
04711                   }
04712               }
04713           }
04714
04715       }
04716
04717       internal class TrueTypeGPOSTable :  ITrueTypeTable
04718       {
04719           public ushort MajorVersion;
04720           public ushort MinorVersion;
04721
04722           public ScriptList ScriptList;
04723           public FeatureList FeatureList;
04724           public LookupList LookupList;
04725
04726           private HashSet<int> KernLookups;
04727
04728           public TrueTypeGPOSTable(Stream fs)
04729           {
04730               long startPosition = fs.Position;
04731
04732               this.MajorVersion = fs.ReadUShort();
04733               this.MinorVersion = fs.ReadUShort();
04734
04735               ushort scriptListOffset = fs.ReadUShort();
04736               ushort featureListOffset = fs.ReadUShort();
04737               ushort lookupListOffset = fs.ReadUShort();
04738
04739               if (this.MinorVersion == 1)
04740               {
04741                   fs.ReadUInt();
04742               }
04743
04744               fs.Seek(startPosition + scriptListOffset, SeekOrigin.Begin);
04745               this.ScriptList = new ScriptList(fs);
04746
04747               fs.Seek(startPosition + featureListOffset, SeekOrigin.Begin);
04748               this.FeatureList = new FeatureList(fs);
04749
04750               fs.Seek(startPosition + lookupListOffset, SeekOrigin.Begin);
04751               this.LookupList = new LookupList(fs);
```

```
04752
04753                 this.KernLookups = new HashSet<int>();
04754
04755                 for (int i = 0; i < this.FeatureList.FeatureRecords.Length; i++)
04756                 {
04757                     if (this.FeatureList.FeatureRecords[i].Tag == "kern")
04758                     {
04759                         for (int j = 0; j <
    this.FeatureList.FeatureRecords[i].LookupListIndices.Length; j++)
04760                         {
04761                             if
    (this.LookupList.LookupTables[this.FeatureList.FeatureRecords[i].LookupListIndices[j]].LookupType ==
    2)
04762                             {
04763
    this.KernLookups.Add(this.FeatureList.FeatureRecords[i].LookupListIndices[j]);
04764                             }
04765                         }
04766                     }
04767                 }
04768
04769             }
04770
04771             public byte[] GetBytes()
04772             {
04773                 using (MemoryStream ms = new MemoryStream())
04774                 {
04775                     ms.WriteUShort(this.MajorVersion);
04776                     ms.WriteUShort(this.MinorVersion);
04777
04778                     int offset = 10;
04779
04780                     if (this.MinorVersion == 1)
04781                     {
04782                         offset += 2;
04783                     }
04784
04785                     ms.WriteUShort((ushort)offset);
04786
04787                     byte[] scriptList = this.ScriptList.GetBytes();
04788                     offset += scriptList.Length;
04789
04790                     ms.WriteUShort((ushort)offset);
04791
04792                     byte[] featureList = this.FeatureList.GetBytes();
04793                     offset += featureList.Length;
04794
04795                     ms.WriteUShort((ushort)offset);
04796
04797                     if (this.MinorVersion == 1)
04798                     {
04799                         ms.WriteUInt(0);
04800                     }
04801
04802                     ms.Write(scriptList, 0, scriptList.Length);
04803                     ms.Write(featureList, 0, featureList.Length);
04804
04805                     byte[] lookupList = this.LookupList.GetBytes();
04806
04807                     ms.Write(lookupList, 0, lookupList.Length);
04808
04809                     return ms.ToArray();
04810                 }
04811             }
04812
04813
04814             public PairKerning GetKerning(int glyph1Index, int glyph2Index)
04815             {
04816                 foreach (int index in this.KernLookups)
04817                 {
04818                     for (int i = 0; i < this.LookupList.LookupTables[index].SubTables.Length; i++)
04819                     {
04820                         if (this.LookupList.LookupTables[index].SubTables[i] is PairPosSubtable
    pairPos)
04821                         {
04822                             int coverageIndex = pairPos.CoverageTable.ContainsGlyph(glyph1Index);
04823
04824                             if (coverageIndex >= 0)
04825                             {
04826                                 return pairPos.GetKerning(coverageIndex, glyph1Index, glyph2Index);
04827                             }
04828                         }
04829                     }
04830                 }
04831
04832                 return null;
04833             }
```

```
04834         }
04835
04836         internal struct TrueTypeTableOffset
04837         {
04838             public uint Checksum;
04839             public uint Offset;
04840             public uint Length;
04841
04842             public TrueTypeTableOffset(uint checksum, uint offset, uint length)
04843             {
04844                 this.Checksum = checksum;
04845                 this.Offset = offset;
04846                 this.Length = length;
04847             }
04848         }
04849
04850         internal struct LongHorFixed
04851         {
04852             public ushort AdvanceWidth;
04853             public short LeftSideBearing;
04854
04855             public LongHorFixed(ushort advanceWidth, short leftSideBearing)
04856             {
04857                 this.AdvanceWidth = advanceWidth;
04858                 this.LeftSideBearing = leftSideBearing;
04859             }
04860         }
04861
04862         internal struct Fixed
04863         {
04864             public int Bits;
04865             public int BitShifts;
04866
04867             public Fixed(int bits, int bitShifts)
04868             {
04869                 this.Bits = bits;
04870                 this.BitShifts = bitShifts;
04871             }
04872         }
04873     }
04874
04875     internal static class ReadUtils
04876     {
04877         public static void WritePascalString(this Stream sr, string val)
04878         {
04879             sr.WriteByte((byte)val.Length);
04880             for (int i = 0; i < val.Length; i++)
04881             {
04882                 sr.WriteByte((byte)(int)val[i]);
04883             }
04884         }
04885
04886         public static string ReadPascalString(this Stream sr)
04887         {
04888             byte length = (byte)sr.ReadByte();
04889             StringBuilder tbr = new StringBuilder(length);
04890             for (int i = 0; i < length; i++)
04891             {
04892                 tbr.Append((char)sr.ReadByte());
04893             }
04894             return tbr.ToString();
04895         }
04896
04897         public static uint ReadUInt(this Stream sr)
04898         {
04899             return ((uint)sr.ReadByte() « 24) | ((uint)sr.ReadByte() « 16) | ((uint)sr.ReadByte() « 8)
      | (uint)sr.ReadByte();
04900         }
04901
04902         public static void WriteUInt(this Stream sr, uint val)
04903         {
04904             sr.Write(new byte[] { (byte)(val » 24), (byte)((val » 16) & 255), (byte)((val » 8) & 255),
      (byte)(val & 255) }, 0, 4);
04905         }
04906
04907         public static int ReadInt(this Stream sr)
04908         {
04909             return (sr.ReadByte() « 24) | (sr.ReadByte() « 16) | (sr.ReadByte() « 8) | sr.ReadByte();
04910         }
04911
04912         public static void WriteInt(this Stream sr, int val)
04913         {
04914             sr.WriteUInt((uint)val);
04915         }
04916
04917         public static ushort ReadUShort(this Stream sr)
04918         {
```

```
04919             return (ushort)(((uint)sr.ReadByte() << 8) | (uint)sr.ReadByte());
04920         }
04921 
04922         public static void WriteUShort(this Stream sr, ushort val)
04923         {
04924             sr.Write(new byte[] { (byte)(val >> 8), (byte)(val & 255) }, 0, 2);
04925         }
04926 
04927         public static short ReadShort(this Stream sr)
04928         {
04929             ushort result = sr.ReadUShort();
04930             short tbr;
04931             if ((result & 0x8000) != 0)
04932             {
04933                 tbr = (short)(result - (1 << 16));
04934             }
04935             else
04936             {
04937                 tbr = (short)result;
04938             }
04939             return tbr;
04940         }
04941 
04942         public static void WriteShort(this Stream sr, short val)
04943         {
04944             sr.WriteUShort((ushort)val);
04945         }
04946 
04947         public static string ReadString(this Stream sr, int length)
04948         {
04949             StringBuilder bld = new StringBuilder(length);
04950             for (int i = 0; i < length; i++)
04951             {
04952                 bld.Append((char)sr.ReadByte());
04953             }
04954             return bld.ToString();
04955         }
04956 
04957         public static TrueTypeFile.Fixed ReadFixed(this Stream sr)
04958         {
04959             return new TrueTypeFile.Fixed(sr.ReadInt(), 16);
04960         }
04961 
04962         public static void WriteFixed(this Stream sr, TrueTypeFile.Fixed val)
04963         {
04964             sr.WriteInt(val.Bits);
04965         }
04966 
04967         public static DateTime ReadDate(this Stream sr)
04968         {
04969             long macTime = sr.ReadUInt() * 0x100000000 + sr.ReadUInt();
04970             return new DateTime(1904, 1, 1).AddTicks(macTime * 1000 * TimeSpan.TicksPerMillisecond);
04971         }
04972 
04973         public static void WriteDate(this Stream sr, DateTime date)
04974         {
04975             long macTime = date.Subtract(new DateTime(1904, 1, 1)).Ticks / 1000 /
      TimeSpan.TicksPerMillisecond;
04976             sr.WriteUInt((uint)(macTime >> 32));
04977             sr.WriteUInt((uint)(macTime & 0xFFFFFFFF));
04978         }
04979     }
04980 
04981 }
```

Index